Oracle Rdb7™

Guide to Database Performance and Tuning, Volume 1 and Volume 2

Release 7.0

Part No. A41747-1

ORACLE®

Guide to Database Performance and Tuning

Release 7.0

Part No. A41747-1

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

The programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, back up, redundancy and other measures to ensure the safe use of such applications if the programs are used for such purposes, and Oracle disclaims liability for any damages caused by such use of the programs.

Contents

Volume I

1 Database Performance Overview

2 Database Performance Analysis Tools

3 Analyzing Performance Factors

4 Adjusting Parameters

Volume II

5 The Query Optimizer

6 Using Oracle Rdb in a VMScluster Environment

7 Tuning Concepts and Methodology

8 Diagnosing a Database Resource Bottleneck

A Oracle Rdb Logical Names and Configuration Parameters

B Oracle Rdb Event-Based Data Tables

C Using RDMS$DEBUG_FLAGS and RDB_DEBUG_FLAGS to Analyze the Query Optimizer

Index

Examples

Figures

Tables

Send Us Your Comments

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

You can send comments to us in the following ways:

- Electronic mail — `nedc_doc@us.oracle.com`
- FAX — `603-897-3334 Attn: Oracle Rdb Documentation`
- Postal service

```
Oracle Corporation
Oracle Rdb Documentation
One Oracle Drive
Nashua, NH  03062
USA
```

If you like, you can use the following questionnaire to give us feedback. (Edit the online release notes file, extract a copy of this questionnaire, and send it to us.)

Name ______________________________ Title ______________________

Company ___________________________ Department _________________

Mailing Address _____________________ Telephone Number ___________

____

Book Title __________________________ Version Number ____________

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?

- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available).

Preface

Oracle Rdb is a general-purpose database management system based on the relational data model.

Purpose of This Manual

This manual describes a database analysis methodology that provides a step-by-step approach to identifying, analyzing, isolating, and solving performance problems. It describes the factors that affect database performance, how to use database analysis tools to examine those factors, and how to adjust database parameters to improve performance.

This manual describes how to use various database tuning tools and utilities to collect and report system, user, and database resource statistics. These tools include the Oracle Rdb Performance Monitor and Oracle Trace software. Oracle Trace software collects event-based data and provides several reports. Oracle Trace can also provide workload input for Oracle Expert for Rdb software.

Oracle Expert for Rdb software allows you to generate an optimal physical design for a database by specifying workload, data volume, and system environment information.

Intended Audience

This manual is intended for experienced database administrators who are responsible for maintaining or improving database performance. You should be familiar with data processing procedures, basic database management concepts and terminology, and the OpenVMS operating system, and be very familiar with Oracle Rdb.

Structure

This manual is divided into two volumes. The chapters in each volume and the appendixes are described in the following table:

Volume I	
Chapter 1	Describes performance factors, introduces the utilities and tools used to analyze performance, and provides a performance analysis methodology.
Chapter 2	Provides an overview of the tools used for analyzing database performance, including the RMU Analyze command, the Performance Monitor, Oracle Rdb logical names, Oracle Trace for OpenVMS software, and Oracle Expert for Rdb software.
Chapter 3	Explains general database performance considerations, such as default parameters, disk I/O, and data distribution. The chapter also provides detailed information on how after-image journaling, locking, and indexed retrieval affect performance.
Chapter 4	Describes how to adjust database and operating system parameters to improve performance.
Volume II	
Chapter 5	Provides an overview of the query optimizer, describes the access strategies the optimizer uses to retrieve data, and explains how you can influence the optimizer.
Chapter 6	Discusses how to configure an Oracle Rdb database in a VMScluster environment.
Chapter 7	Describes database tuning and provides a tuning methodology to help you determine what and when to tune.
Chapter 8	Provides a series of decision trees to help you diagnose database resource bottlenecks.
Appendix A	Provides a detailed description of the Oracle Rdb logical names and configuration parameters.
Appendix B	Describes the Oracle Rdb event-data tables provided by Oracle Trace for OpenVMS software.
Appendix C	Describes how to use the RDMS$DEBUG_FLAGS logical name and the RDB_DEBUG_FLAGS configuration parameter to examine optimizer retrieval strategies, query execution, and query cost.

Related Manuals

For more information on Oracle Rdb, see the other manuals in this documentation set, especially the following:

- *Oracle Rdb7 Introduction to SQL*
- *Oracle Rdb7 Guide to SQL Programming*
- *Oracle Rdb7 SQL Reference Manual*
- *Oracle RMU Reference Manual*
- *Oracle Rdb7 Guide to Database Design and Definition*
- *Oracle Rdb7 Guide to Database Maintenance*
- *Oracle Rdb7 Installation and Configuration Guide*
- *Oracle Rdb7 Release Notes*

The *Oracle Rdb7 Release Notes* list all the manuals in the Oracle Rdb documentation set.

The following documentation sets provide related information:

- The Oracle Expert for Rdb documentation set
- The Oracle Trace for OpenVMS documentation set
- The Oracle CDD/Repository documentation set
- The documentation set for your operating system

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press Return at the end of a line of input.

Often in examples the prompts are not shown. Generally, they are shown where it is important to depict an interactive sequence exactly; otherwise, they are omitted in order to focus full attention on the statements or commands themselves.

Discussions in this manual that refer to VMScluster environments apply to both VAXcluster systems that include only VAX nodes and VMScluster systems that include at least one Alpha node, unless indicated otherwise.

In this manual, OpenVMS means the OpenVMS Alpha operating system, the OpenVMS VAX operating system, and the VAX VMS operating system.

This manual uses icons to identify information that is specific to an operating system or platform. Where material pertains to more than one platform or operating system, combination icons or generic icons are used. For example:

Digital UNIX — This icon denotes the beginning of information specific to the Digital UNIX operating system.

OpenVMS VAX OpenVMS Alpha — This icon combination denotes the beginning of information specific to both the OpenVMS VAX and OpenVMS Alpha operating systems.

♦ — The diamond symbol denotes the end of a section of information specific to an operating system or platform.

The following conventions are also used in this manual:

Ctrl/x — This symbol in examples tells you to press the Ctrl (control) key and hold it down while pressing the specified letter key.

Return — This symbol in examples indicates the Return key.

Tab — This symbol in examples indicates the Tab key.

.
.
. — Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.

. . . — Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.

e, f, t — Index entries in the printed manual may have a lowercase e, f, or t following the page number; the e, f, or t is a reference to the example, figure, or table, respectively, on that page.

< > — Angle brackets enclose user-supplied names.

[ ] — Brackets enclose optional clauses from which you can choose one or none.

$ — The dollar sign represents the DIGITAL Command Language prompt in OpenVMS and the Bourne shell prompt in Digital UNIX.

Volume I

1 Database Performance Overview

After a database has been implemented and put into production, daily use of that database continually tests the database design, both logical and physical. Eventually, users may report that database response time has degraded. For example, a standard report that originally was generated in seconds may now take minutes. Performance degradation may occur gradually or it can happen overnight; it may be chronic or it may be intermittent; it may affect all users and applications or only some. This manual will help you diagnose and solve database performance problems.

This chapter introduces the concepts of database performance and tuning. It briefly describes the factors that affect performance and the tools you can use to isolate a problem. Section 1.4 describes a performance analysis methodology that provides guidelines for logically finding the source of a performance problem.

Chapter 7 and Chapter 8 provide information that supplements the earlier chapters. These two chapters further define the concept of tuning and explore how tuning a system, database, and application can affect database performance. Chapter 8 presents a series of decision trees to aid in identifying, analyzing, isolating, and solving a performance problem, and in monitoring the resulting solution. Oracle Corporation recommends that you read and understand the earlier material before you read Chapter 7 and Chapter 8.

1.1 Performance and Tuning

In the broadest sense, database performance is a measure of user satisfaction. It is up to you to establish realistic performance expectations. Users must be aware that response time can vary due to many factors, some of which may be beyond your control because of resource limitations. Optimum performance for a given database and system configuration is defined as the best possible response time for the most commonly executed database operations.

This manual addresses factors affecting database performance that you can analyze and adjust. By monitoring and evaluating database performance characteristics such as locking, data distribution, and I/O, you may determine that performance can be improved by adjusting one or several of the factors discussed in Section 1.2.

When you adjust a factor that affects database performance, you are tuning a database. There is a distinction between tuning and troubleshooting. Troubleshooting, which is discussed in the *Oracle Rdb7 Guide to Database Maintenance*, involves analyzing a bugcheck dump file that results from a software error. Tuning involves adjusting parameters that affect how efficiently data is read or written to a database to achieve optimum performance.

A database is tuned once during the initial design phase and may require subsequent tuning after implementation to optimize performance. This subsequent tuning may be in response to one or more of the following occurrences:

- User complaints
- Database growth
- Changes that alter the physical characteristics of the database
- Performance monitoring
- Addition or subtraction of resources
- New applications or workload changes
- Additional users

During the initial physical design, a database is created without the benefit of any performance data. The database creator uses the guidelines described in the *Oracle Rdb7 Guide to Database Design and Definition* to arrive at the best possible physical design. This manual presumes that you already have a database up and running and need to refine an existing design. For more information on tuning, refer to Chapter 7.

1.2 Performance Factors

Performance degradation is a common problem. However, the source of a performance problem is often not obvious. Some general areas that can be sources of database performance problems are shown in Table 1–1.

Table 1–1 Performance Problem Areas

Problem Area	What to Investigate
System resources	Do you have enough memory, a sufficient number of disks, and enough CPU power for the number of users you support?
Memory management	Are your parameter values optimally tuned for user processes accessing your database application?
Operating system parameters	Are system parameter values set properly?
Process parameters	Are user account quotas set properly?
Logical database design	Is it taking too long to find or update certain rows?
Physical database design	Has the logical database design been implemented efficiently? • Parameters. Have you altered the default Oracle Rdb parameters in accordance with site-specific needs? • Row placement. Are there too many fragmented or displaced rows? Are the pages too full? Are there too many hash bucket overflows? Are many data rows displaced on different pages from where their hash buckets are located? • Database space. Is your database getting too full? Are there many extents for any storage areas? Have you outgrown initial space allocation sizes? • Locking problems. Are too many programs trying to update or read the same rows at the same time?
Application design	Do your SQL precompiler and SQL module language programs use the database efficiently, and are tables reserved intelligently? Are transactions designed to allow maximum concurrency?

Once you determine the problem area, you can begin evaluating factors that affect database performance within that area. The remainder of this section briefly describes each potential problem area and indicates where you can find additional information.

1.2.1 System Resources and Memory Management

Insufficient CPU power, storage, or memory can result in unacceptable response times. Refer to the following sources to determine if your performance problems are caused by insufficient system resources:

- OpenVMS VAX OpenVMS Alpha Section 1.3.1 of this manual describes operating system utilities and other system performance utilities that can help you identify system-related problems. ♦
- Section 8.3 describes how to evaluate CPU resources.
- OpenVMS VAX OpenVMS Alpha The OpenVMS documentation set has information on procedures that can identify performance problems. It also provides procedures to follow to achieve optimum performance for your system configuration and for the workload you need to manage. ♦

One aspect of tuning overall system performance involves a careful analysis of the memory management of the system. Refer to Section 4.4.3, Tuning Working Set Adjustment Parameters, and Section 8.2, Analyzing Memory Resources, for more information.

1.2.2 OpenVMS System Parameters and Process Parameters

OpenVMS VAX OpenVMS Alpha Although you may have sufficient system resources, good database design, and properly set Oracle Rdb parameters, database performance is still subject to the performance of the system itself. During the installation of Oracle Rdb you should have set OpenVMS parameter values and process account quotas according to the guidelines in the *Oracle Rdb7 Installation and Configuration Guide*.

When tuning is required, you should select a very small number of parameters for change, based on a careful analysis of the observed behavior, or based on the results of running a system performance monitor. Tools for analyzing and monitoring your system are described in Section 1.3.1. The parameters are usually either system parameters or entries in the user authorization file (UAF) that affect particular users. You modify system parameters using the AUTOGEN command procedure. One AUTOGEN feature is that it automatically adjusts associated parameters to any changes you make. To control the values in the UAF, you use the OpenVMS Authorize utility (AUTHORIZE). See the OpenVMS documentation set for detailed information on system parameters, the UAF file, AUTOGEN, and the Authorize utility.

Section 4.4.2 and Section 4.4.4 discuss setting system parameters and user account quotas in more detail. ♦

1.2.3 Database Design

The logical design of your database is the foundation upon which all other performance factors depend. If the logical design of your database is inadequate, adjusting other parameters, such as operating system parameters and Oracle Rdb parameters, cannot optimize performance. A poor logical design requires substantial effort to re-analyze, redesign, restructure, and reload the database. As a starting point for improving database performance, you should gain a thorough understanding of your data and be familiar with the design concepts presented in the *Oracle Rdb7 Guide to Database Design and Definition*.

Refining the physical design of a database (tuning) involves analyzing the factors that can affect data storage and retrieval, and then adjusting the parameters that control those factors. Analyzing and adjusting parameters are discussed in Chapters 3 and 4, respectively. Additional information is available in Chapter 8, Diagnosing a Database Resource Bottleneck. You should also consider using Oracle Trace and Oracle Expert for Rdb[1]. Refer to Section 1.3.1 for a brief description of these two products.

1.2.4 Application Design

The final element that can adversely affect database performance is application design. Use the following general guidelines in database programming to optimize performance and reduce contention:

- Keep transactions short to reduce locking and to keep the snapshot file from extending (if enabled).
- Keep transactions simple by dividing a complex query into several simpler queries and by using views to standardize program access. These steps will:
  - Reduce query optimization overhead
  - Reduce potential programming and maintenance problems
- Use SHARED READ with the RESERVING clause to minimize locking and promote concurrent access.
- Avoid terminal I/O within a transaction to prevent intermittent locking problems.
- Help the optimizer. Refer to Section 5.8 for information.
- Use dbkeys, where possible, to access data rows directly. This avoids use of an index, which minimizes locking, and also reduces I/O.

[1] Oracle Trace and Oracle Expert for Rdb are available on OpenVMS systems only.

1.2.5 Possible Performance-Related Changes

This section lists some of the changes you can make to a system, or to a database, that can impact performance.

OpenVMS VAX OpenVMS Alpha

Possible system changes include adjusting OpenVMS parameters such as LOCKIDTBL, REHASHTBL, GBLPAGES, GBLSECTIONS, MAXBUF, VIRTUALPAGECNT, and DEADLOCK_WAIT for Oracle Rdb applications to determine optimum settings.

Possible process changes include adjusting user account parameters such as ENQLM, WSQUO, WSDEF, WSMAX, FILLM, BYTLM, ASTLM, DIOLM, and BIOLM for Oracle Rdb applications to determine optimum settings. ♦

Possible database changes include the following options:

- Data distribution

  Redistribute database files among your disk devices.

  Redistribute rows among storage areas to achieve a more even distribution. Also, horizontally partition rows across storage areas to group commonly accessed rows.

  Cluster two or more tables within the same mixed storage area so rows from these tables are read into a buffer together. If clustering is not possible, consider implementing shadow pages.

  Control data redundancy to improve retrieval performance (require fewer joins) at the possible expense of update performance.

- Indexes

  Add hashed indexes to mixed storage areas.

  Set index characteristics (node size, percent fill, and usage options) based on predominant database use (query or update transactions).

  Define indexes after rows are loaded into the database; sort rows by primary key before loading them into the database.

- Locking

  Disable adjustable lock granularity when contention for rows in a logical area is very high and CPU time is a problem. Also consider reducing row lock conflicts further by changing read/write storage areas that are never updated to read-only storage areas. Use data compression for those tables that are not updated frequently.

  Optimize transaction scope as a means of reducing potential lock conflicts with other database users.

  Enable or disable snapshots to reduce lock conflicts between update and retrieval users (also consider using deferred snapshot files).

- I/O

  Optimize buffer size, number of buffers, and buffer pools for the dominant transaction type for the database.

  Depending on the dominant transaction type, enable global buffers or fast commit processing or both.

- Database parameters

  Optimize page size for different storage areas to avoid row fragmentation.

  Experiment with page allocation ranges and page block sizes to determine which configurations provide the most even row distribution results.

  Set SPAM threshold values and the data page interval between SPAM pages for uniform and mixed storage areas.

  Perform an IMPORT/EXPORT operation to eliminate extensions in mixed storage areas.

1.3 Performance Utilities and Tools

Several utilities and tools are available to help you evaluate the performance of your database environment. These include OpenVMS utilities, Oracle RMU commands, and Oracle Rdb logical names and configuration parameters. Section 1.3.1 through Section 1.3.3 briefly introduce the various tools and utilities and direct you to more information.

1.3.1 Operating System Utilities

OpenVMS VAX OpenVMS Alpha

Database performance is closely related to overall operating system performance because of the great impact of a database management system on an operating system.

The OpenVMS operating system provides the following utilities to help you evaluate the performance of your database within the overall context of the OpenVMS operating system environment:

- The DCL MONITOR command shows continuous system resource usage and statistics.
- The DCL SHOW commands display a snapshot of system resource usage.
- The DCL DUMP command enables you to display or print the contents of files or volumes in ASCII, decimal, hexadecimal, or octal representation.
- SYSGEN shows current OpenVMS system parameters.

- AUTOGEN enables you to modify OpenVMS system parameters.
- AUTHORIZE enables you to show and modify user quota parameters.

Refer to the OpenVMS documentation set for further information on these OpenVMS utilities and commands.

Oracle products that are useful for monitoring system performance include:

- Oracle Trace software

  Oracle Trace is a product that collects and reports on event-based data gathered from any combination of OpenVMS layered products and Oracle Rdb applications. Oracle Rdb has many predefined events that occur during run time. An event can have a start and an end (beginning of a transaction to commit) or it can simply occur (attach to a database). Oracle Trace allows data items to be associated with these predefined Oracle Rdb events. These items can be standard resource utilization items or items specific to applications using Oracle Rdb. For example, items can be information about the event itself, such as the name of the event or in what procedure the event is occurring.

  Items can also include process statistics and performance information, such as working set size at the time the event occurs. If an application is instrumented with Oracle Trace calls (described in Section 2.4.1), you can use Oracle Trace to collect event data from the application. This event data can be useful for many different purposes, including:

  - Tuning the performance of applications
  - Planning hardware resources (capacity planning)
  - Tuning the performance of databases
  - Debugging applications
  - Logging errors
  - Importing application performance data into Oracle Expert for Rdb

  Oracle Trace differs from other collectors in that it is event-based, whereas most other collectors are timer-based. An **event-based collector** gathers data at predefined locations in your program code when that code is executed. **Timer-based collectors** perform data collection at specified time intervals. Advantages of event-based collectors include:

  - Provide an easy way to collect and report on the actual resources used by certain events in applications; for example, to generate a specific report or transaction.

  - Determine the actual frequency of the execution of events, rather than an average or estimated frequency, such as how often a transaction is executed.

  Oracle Trace does not attempt to analyze or modify the performance of an application or database. Its function is to collect data requested by users and to provide reports based on that data. Interpreting these reports is the responsibility of the user or of other layered products. See Section 2.4 for information on how you can use Oracle Trace with Oracle Rdb applications. Refer to the Oracle Trace documentation for additional information on using Oracle Trace.

- Oracle Expert for Rdb software

  Oracle Expert for Rdb is a software tool that helps you design and implement a database. Oracle Expert for Rdb contains extensive tuning rules that are based on physical database design principles and database management internals, such as the Oracle Rdb query optimizer.

  To develop a design, Oracle Expert for Rdb imports the logical design of the database, table and index cardinalities, and transaction workload information. It requires input of the workload priorities and database environment information. A report is generated that documents and explains why this particular design was chosen. Workload data collected by Oracle Trace can be directly imported to Oracle Expert for Rdb.

  After the information is analyzed, Oracle Expert for Rdb produces command files that are used to create the database. Oracle Expert for Rdb can also generate procedures for loading and unloading the database.

  You can also use Oracle Trace and Oracle Expert for Rdb to tune an existing database. Use Oracle Trace to collect data on the running system and import it into Oracle Expert for Rdb. Then use Oracle Expert for Rdb to unload the existing database, make design changes, and reload the database. Refer to the *Oracle Expert for Rdb User's Guide* for more information. ♦

1.3.2 Oracle RMU Commands

Oracle RMU, the Oracle Rdb management utility, enables you to display information about Oracle Rdb databases. Oracle RMU commands that are particularly useful for analyzing database performance include: RMU Analyze, RMU Show, and RMU Dump.

RMU Analyze

A major cause of database performance problems is poor row storage. You can use the RMU Analyze command to monitor and evaluate database space usage. A regular analysis of database space usage can help you spot potential problems early and let you begin to look for solutions before the problems become acute. Chapter 2 introduces the RMU Analyze command and its qualifiers. Specific examples of RMU Analyze and its output are located in Chapters 3 and 4 of this manual.

RMU Show

Oracle Rdb maintains statistics for various database activities on a database-wide basis. You can use these statistics to evaluate the efficiency of a program's database access in terms of I/O operations for each database area. You can also use the statistics to evaluate your index node, to evaluate overall database access patterns, and to evaluate locking statistics.

Chapter 2 describes how to use the RMU Show Statistics command to invoke the Performance Monitor to display database statistics. The Performance Monitor screens are described throughout this manual.

The RMU Show Locks command displays current information about process locks for all active databases on a specific node. By determining the type of lock held by a process or list of processes, you can identify which process is blocking other processes. The RMU Show Locks command is described in Section 3.8.1.1.

RMU Dump

In evaluating performance and data distribution problems, especially in the area of row placement, you may find it useful to examine the on-disk contents of your database. You may also want to review internal root file information and ID numbers for database storage areas and logical areas. You can also display storage area and logical area information to inspect space area management (SPAM) pages, area inventory pages (AIPs), area bit maps (ABMs), and data pages where your index structures and data rows are located. Refer to the *Oracle Rdb7 Guide to Database Maintenance* for information on using the RMU Dump command to display and interpret the contents of your database files.

Refer to the *Oracle RMU Reference Manual* for a full explanation of these Oracle RMU commands.

1.3.3 Oracle Rdb Logical Names and Configuration Parameters

Oracle Rdb provides many logical names and configuration parameters you can define that can improve performance by giving process-level control over several parameters. For example, Oracle Rdb logical names and configuration parameters enable you to relocate or pre-extend the .ruj file or temporarily override the default number of user-allocated buffers at run time. Other Oracle Rdb logical names and configuration parameters help you examine optimizer strategy.

Table 2–6 lists all the Oracle Rdb logical names and configuration parameters, and provides a brief functional description of each one. Appendix A describes each logical name and configuration parameter in detail.

1.4 Performance Analysis Methodology

Before you start to evaluate the performance of your database, eliminate other possible causes of poor database performance, such as insufficient or faulty hardware or operating system resources. Avoid devoting a lot of time to evaluating the database environment unless you are sure the problem lies there and not in hardware or operating system resources and quotas. Remember that any change in your system configuration or a dramatic change in system workload can affect overall system performance.

Most database performance considerations depend on the specific attributes of your application. There are no hard and fast rules about design, analysis, tuning, or programming. There is no one set of answers to any database performance problem. You must evaluate database performance according to your database's intended and actual use. Performance problems often occur when a database design does not match the eventual use of that database.

The remainder of this section presents a series of recommendations for analyzing database performance. You should also refer to the decision trees in Chapter 8 for detailed, step-by-step help in evaluating specific problems.

1.4.1 Tuning Guidelines

The first step in analyzing database performance is determining if a problem really exists. Complaints about performance may arise from unrealistic user expectations. If you determine that a problem does exist, ask yourself the following general questions:

- When did the problem first appear?
- Is the problematic behavior intermittent or consistent?
  - Does it occur more often at a certain time of day?

  - Does it occur with a particular type of transaction?
  - Does one person or program experience the problem more often than others?
  - When did the problem last occur? What else was happening on the system?
- Were any changes made just before the problem first arose? Has new software or hardware been installed recently?
- Have workload volumes or other environmental factors changed recently?
- Are third-party products being used with the database?

The answers to these questions can suggest when and where to begin gathering information.

1.4.2 Establishing a Context to Interpret Tuning Results

To analyze the performance of your database, you should establish a context within which to interpret the results of your evaluation. Set a goal and measure your progress toward it. For example, set a goal of improving average response time by 10 percent. Measure the effect of a performance improvement, such as enabling page locking to remove the overhead of record locks for partitioned applications.

If you do not know what you want to achieve in measurable terms when you begin evaluating and improving database performance, you will not know if the performance gains you make solve your problem. When you measure the results of each change, you then know how much improvement resulted from each change you made. Keep a log of the changes you make and the percent increase in performance that results so you can see if the same areas repeatedly cause problems.

Try to tune when a performance problem is most evident. This will make it easier to identify the cause of the problem.

1.4.3 Using a Test Database

Use a test database to help you evaluate the performance of your database and the programs that access it. You should try to create a realistic testing environment, not only for performance testing, but also for testing database modifications before you implement them in the production database.

If your database is too large to create an exact copy, create a representative fraction, a tenth for example. Then load the test database with the correct proportion of test data to about the same fullness percentage as your production database. Make sure the page sizes match. Your results may not be exactly the same because the storage hashing algorithm may produce some slightly different results due to the different page count, but with a database scaled down by a factor of 10, the results should be similar.

Verify that you can create the same problem on your test database. If necessary, run your production programs against it. Set up a complete test directory structure that parallels your production directory structure, if you have room. You may want to dedicate a disk or two to this effort. An investment in an adequate testing environment can pay off quickly in improvements to the production database, and it will reduce the risk of tuning directly on the production database.

1.4.4 Making Changes

Two rules to follow when changing parameters that affect database performance are:

- Make one change at a time.
- Make changes in order of difficulty.

1.4.4.1 Make Single Changes

Begin to make changes to the test database and to the programs that run against it, implementing one change at a time. Note the effect of each change, and evaluate the impact of the change to the database environment. If the change makes performance worse, restore the database environment to the original state.

As you make modifications to the test database and verify them, construct a plan for implementing those modifications in the production database.

1.4.4.2 Make Changes in Order of Difficulty

You should make performance-related changes to your database in order of their difficulty to implement.

- You should make the most easily implemented changes first. For example, you can make some changes on line (with users attached to the database and transactions in progress) that go into effect immediately, such as after-image journal file allocation and extension.
- You can make some changes on line whose effect is deferred until a new transaction occurs or a user detaches and reattaches to the database. An example of this type of change is setting the read-only flag that indicates whether a storage area is read-only or read/write.
- You can make changes off line (no users attached to the database) when they do not require unloading and reloading the database, such as changing the number of users allowed to attach to the database.

- You can make changes off line that require the database to be unloaded and reloaded, such as adding storage areas and reorganizing storage areas.

Section 4.1 describes how to adjust database parameters. Most performance problems that relate to changing the specifications for particular database-wide parameters and storage area parameters can be solved in one of the following ways:

- By using an SQL ALTER DATABASE statement and making the appropriate database-wide changes
- By using an SQL ALTER DATABASE statement and defining a new storage area, modifying the associated storage map statements, and automatically moving the rows for specified tables into the new storage areas

See the *Oracle Rdb7 Guide to Database Maintenance* for a complete description of the database modifications you can make on line and off line and without exporting and importing your database. The *Oracle Rdb7 Guide to Database Maintenance* also describes procedures you can develop to export and import your database using the SQL EXPORT and IMPORT statements.

1.4.5 Sample Procedure for Performance Evaluation

This section outlines a sample procedure for evaluating the performance of an online database application.

1. Use the SQL EXPORT and IMPORT statements to move the database to a new disk location and to move the data repository (Oracle CDD/Repository[1]) information to a new repository directory.
2. Identify and isolate sections of application program code that access the database.
3. Create program test modules that reproduce the actual operation of the application.

OpenVMS VAX OpenVMS Alpha

4. Use OpenVMS Run-Time Library calls LIB$INIT_TIMER and LIB$SHOW_TIMER or Oracle Trace to produce OpenVMS operating system statistics, such as elapsed time and CPU time, for the program test modules. ♦
5. Use the Performance Monitor to monitor Oracle Rdb statistics, such as the number of database pages read, locking activity, and I/O statistics.
6. Establish a workload procedure (run the program modules at certain intervals, or create batch processes to simulate online users).

[1] On OpenVMS systems only

7. Make changes to the test database and compare statistical output before and after the changes.

1.4.6 Cluster Performance Considerations

Database performance in a cluster environment may be affected by how and where you place your database files on disks. If you are using Oracle Rdb in a large cluster environment, for example, you might want to place your database root file on a separate disk because I/O activity to the root file increases in a cluster environment.

Also, you may want to distribute large or heavily used storage areas across different disks by changing device specifications using the RMU Restore or RMU Move_Area command. You may want to experiment with various disk access paths to your files. For example, you may discover that database files on HSC disks can be accessed faster than database files on MASSBUS or UNIBUS disks because of HSC seek and rotational optimizations.

OpenVMS VAX ≡ OpenVMS Alpha ≡ Refer to Chapter 6 for more information on Oracle Rdb performance in a VMScluster environment. ♦

2

Database Performance Analysis Tools

This chapter describes the tools you can use to analyze the performance of an Oracle Rdb database. These tools include:

- The RMU Analyze command
- The Performance Monitor
- Oracle Rdb logical names and configuration parameters
- OpenVMS VAX OpenVMS Alpha Oracle Trace for OpenVMS
- Oracle Expert for Rdb ♦

This chapter provides an overview of the Oracle RMU commands and the Oracle Rdb logical names and configuration parameters; you can find specific directions for using each tool in the appropriate section of this manual. For example, the RMU Analyze Indexes command is described in Section 3.9.5. Oracle Trace is covered in Section 2.4.

2.1 RMU Analyze Command

You can use the RMU Analyze command to examine the following database characteristics:

- The space the database is using, how space is utilized in logical areas within storage areas (at the page level), and the amount of free space on each page
- The current database parameters that support your database definitions such as space allocation, page size, SPAM thresholds and intervals, page format, row compression, and general row placement information; and how efficiently these parameters work
- The number and type of records the database currently holds, their location on the page, and statistics on row fragmentation and row compression

- The structure of the indexes, number of levels, number of duplicate nodes, number of unique keys, and detailed information on each index and node record such as the actual dbkey, the level, the size, the key length, and the actual key value
- Index and data record placement for a specified index on a table, the number of index records accessed to reach a data row, the maximum number of pages traversed to reach a data row, and the number of database buffers that are required to access a data row
- Histogram (frequency distribution) charts for each of the following parameters:
  - The number of records accessed
  - The number of pages traversed
  - The number of buffers used

  Detailed information on the frequency distribution of data rows includes:
  - The number of dbkeys needed to access the data row
  - The number of pages accessed to reach a data row
  - Whether or not the data row and index structure would be in the buffer
  - The key length
  - The actual key value

The RMU Analyze command can help you with the following tasks:

- Resetting database parameters using the SQL ALTER DATABASE statement when fragmentation and page overflow occur
- Developing procedures for exporting and importing a database
- Defining indexes
- Adding new storage areas and partitioning rows from one or more tables across these new storage areas to help alleviate I/O bottlenecks
- Adding new storage areas to reorganize rows from old storage areas based on modifications made to a storage map

You can write the results of the RMU Analyze command into a binary file by using the Binary_Output qualifier.

To use the RMU Analyze command, you must have the RMU privilege RMU$ANALYZE or the OpenVMS privileges SYSPRV or BYPASS.

2.1.1 RMU Analyze Command Qualifiers

The RMU Analyze command output shows how full the database is, whether or not rows are fragmented, how rows are distributed, the type of record on the page, and the I/O path length to reach a data row. Such information helps you see how your database is evolving and decide how to improve its performance.

This section provides a general description of the information you can gather using the RMU Analyze command with the four qualifiers shown in Table 2–1. Table 2–1 also indicates where you can find more information and examples of each qualifier.

Table 2–1 RMU Analyze Command Qualifiers

Qualifier	Reference
Areas	Section 4.2.1.1
Lareas	Section 4.2.1.2
Indexes	Section 3.9.5.1
Placement	Section 4.3.1.1 through Section 4.3.1.3

You can vary the level of detail output by each qualifier by using the Option={Normal | Full | Debug} qualifier.

- Normal

  Output includes only summary information. This is the default.

- Full

  Output includes histograms and summary information.

- Debug

  Output includes internal information about the data, as well as histograms and summary information.

You can improve database performance by paying particular attention to the display elements listed under each of the following RMU Analyze command qualifiers:

- Areas
  - Page fullness for the storage area

    Examine the shape of the storage area page space use by page histogram. If many pages are less than 60 percent full, you may be performing many random deletions, or your free page space may be too small to hold a row of typical size. For example, if the page size is 2 blocks (1024 bytes) and a typical row is 550 bytes (after compression), Oracle Rdb could store 1 row per page. Each page would then have 474 bytes of unused space. If you wanted to store 7 rows, it would take 7 pages or 7168 bytes, 3318 bytes of which would be unused ($7 * 474 = 3318$).

    However, if the page size were 4096 bytes, Oracle Rdb could fit all 7 rows on a single page. Only 246 bytes would be unused compared to the 3318 unused bytes with the 1024-byte page size.

  - SPAM count

    The number of SPAM pages approximates the maximum number of I/O operations necessary to find free space when the storage area is filled to near capacity.

  - Overflow

    Overflow can be caused by row fragmentation and hash bucket overflows. Row fragmentation can be due to underestimating the page size for the storage area. It is also caused by not allowing sufficient space for storing the uncompressed rows on a page if the compression characteristic is changed from compressed to uncompressed. Overflow of hashed index records (hash buckets and duplicate node records) and data rows occurs if the page size is set too small relative to all the records that must fit on the page. This may result from not estimating the distribution of duplicate records, or from a low size estimate of a hash bucket on the page, especially when the PLACEMENT VIA INDEX option is used to store both data rows and hash structures on the same page.

- Lareas

  - Fragmentation

    Any fragmented rows are indicated in the summary portion of the output for the RMU Analyze command for the Area and Lareas qualifiers. The information displayed indicates the number of rows that are fragmented, the number of fragments in each row, and the size in bytes of each fragmented row. If many rows are fragmented, performance drops.

  - Page space use by page for logical areas

Examine the shape of the logical area page space use by page histogram. This histogram summarizes the percentage of page space used by data in a specific logical area versus the number of pages that contain data. The shape of the histogram can tell you how rows are distributed among the pages, and can also tell you something about how the database is changing.

  - Percent of maximum record length by record for logical areas

    Examine the shape of the logical area percent of maximum record length by record histogram generated by the Option=Full qualifier. This histogram summarizes for a logical area each record's percent of maximum size (uncompressed) for its record type. The shape of the histogram can reveal the distribution of record lengths in a logical area.

- Indexes
  - Index levels

    Measures the cost of retrieval using the number of I/O operations.

  - The number and size of index nodes

    Measures the storage overhead involved in using the index.

  - The number and size of duplicate indexes

    Measures the efficiency of the index. A unique index is faster than one with many duplicate values that can cause overflow to adjacent pages. Overflow can occur as pages fill up with index records and spill over to adjacent pages because the page size may be too small.

- Placement
  - Maximum and average path length

    Indicates the maximum and average number of index records accessed to reach a data row.

  - Maximum I/O path length

    Indicates the maximum number of pages traversed to reach a data row.

  - Minimum I/O path length

    Indicates the number of buffers required to access a data row for a given buffer size.

  - Dbkey path length by frequency

    Examine the shape of the dbkey path length by frequency histogram. This histogram summarizes the number of index records accessed to reach the data rows. For a specific index, the shape of the histogram provides a frequency distribution of the dbkey path lengths for all data rows in the table for which the index was defined.

  - Maximum I/O path length by frequency

    Examine the shape of the maximum I/O path length by frequency histogram. This histogram summarizes the total number of pages traversed to reach a data row. For a specific index, the shape of the histogram provides a frequency distribution of the total number of pages traversed to reach each data row in the table for which the index was defined.

  - Minimum I/O path length by frequency

    Examine the shape of the minimum I/O path length by frequency histogram. This histogram summarizes, when the buffer size is considered, the number of buffers required to access data rows. For a specific index, the shape of the histogram provides a frequency distribution of the number of buffers used to access data rows in the table for which the index was defined.

  - Detailed information on the distribution of data rows on data pages

    Indicates the number of dbkeys needed to reach a specific data row on the data page, the maximum and minimum I/O path length to reach the data row, the length of each key, and the specific key for the data row.

The RMU Analyze command, used with the Areas and Lareas qualifiers, does not start a transaction, but does lock a physical area with a concurrent read lock while that area is being analyzed. Analysis is performed on one database storage area at a time, with results reported by storage area and logical area. Oracle Rdb maintains separate logical areas for each table's data, for each table's indexes, and one logical area per database for segmented string data. Because the RMU Analyze command does not use record locking or the snapshot file, it allows concurrent database activity.

Generally, all other transaction types (read-only, read/write shared read, read/write protected read, read/write shared write, and read/write protected write) that may need to read or update an area of the database will not conflict with the RMU Analyze command when the Areas and Lareas qualifiers are specified. Snapshot files do not need to be enabled in order to read information from a storage area. Note that the results of the RMU Analyze command may change over time because of ongoing database activity.

When you use either the RMU Analyze Indexes command or the RMU Analyze Placement command, a read-only transaction is started. Snapshot files must be enabled in order to gather information. This permits other database activity to occur while you are using either the RMU Analyze Indexes command or the RMU Analyze Placement command. The only conflicts that might occur are with exclusive and batch-update transactions that try to access the same physical area that is undergoing index or placement analysis. This is because neither transaction writes before-images to the snapshot file.

2.1.1.1 Excluding Oracle Rdb Information from RMU Analyze Command Output

It is possible to exclude Oracle Rdb information from RMU Analyze, RMU Analyze Indexes, and RMU Analyze Placement command output.

You can specify the Exclude=System_Records or the Exclude=Metadata qualifier with the RMU Analyze command. When you specify the Exclude=System_Records qualifier, information on the RDB$SYSTEM_RECORD logical area is excluded from the RMU Analyze output. When you specify the Exclude=Metadata qualifier, information on all the Oracle Rdb logical areas (for example, the RDB$SYSTEM_RECORD and the RDBVMS$COLLATIONS_NDX logical areas) is excluded from the RMU Analyze output.

When you specify the Exclude=Metadata qualifier with either the RMU Analyze Indexes or RMU Analyze Placement command, information on the Oracle Rdb indexes (for example, the RDB$NDX_REL_NAME_NDX and RDBVMS$COLLATIONS_NDX indexes) is excluded from the RMU Analyze Indexes and RMU Analyze Placement output.

Data is accumulated for the logical areas excluded with the Exclude qualifier, but the data is excluded from the command output.

2.1.2 Creating a Binary Output File for Further Analysis

A very useful option of the RMU Analyze command is to output the results into a binary file that is either a fixed-length record binary file (.unl) or an Oracle CDD/Repository[1] compatible record definition file (.rrd) for further analysis such as:

- Loading the summary results into an Oracle Rdb database using the RMU Load Rms_Record_Def command for use by a user-written management application or procedure
- Providing access to the results for a user-written program

[1] On OpenVMS systems only

The binary output file can be created when you specify the Binary_Output qualifier with any of the following commands:

- RMU Analyze (with either the Areas and Lareas qualifiers)
- RMU Analyze Indexes
- RMU Analyze Placement

You can create the fixed-length record binary file with a .unl file type by specifying the Binary_Output= (File = file-spec) qualifier, as in Example 2–1.

Example 2–1 Creating a Fixed-Length Record Binary File

```
$ RMU/ANALYZE/BINARY_OUTPUT=FILE=ANALYZE_OUT1
```

You can use the Binary_Output= (Record_Definition = file-spec) qualifier to create an Oracle CDD/Repository compatible record definition binary file (with an .rrd file type), as in Example 2–2.

Example 2–2 Creating an Oracle CDD/Repository Compatible Record Definition Binary File

```
$ RMU/ANALYZE/BINARY_OUTPUT=RECORD_DEFINITION=ANALYZE_OUT2
```

You can create both binary output files on the same command line by enclosing them within parentheses and separating them with a comma, as in Example 2–3.

Example 2–3 Creating Both a Fixed-Length Record File and an Oracle CDD/Repository Compatible Record Definition File

```
$ RMU/ANALYZE/BINARY_OUTPUT=(FILE=ANALYZE_OUT1,RECORD_DEFINITION=ANALYZE_OUT2)
```

Note that the default is Nobinary_Output and that no binary output file is created.

OpenVMS VAX OpenVMS Alpha

The RMU Analyze command in Example 2–4 outputs the results into an RMS record definition file called DB.RRD that is compatible with the data repository.

Example 2–4 Using RMU Analyze to Create an Oracle CDD/Repository Compatible Record Definition File

```
$ RMU/ANALYZE/BINARY_OUTPUT=RECORD_DEFINITION=DB.RRD mf_personnel
$!
$! Display the DB.RRD file created by the previous command:
$ TYPE DB.RRD
DEFINE FIELD RMU$DATE DATATYPE IS DATE.
DEFINE FIELD RMU$AREA_NAME DATATYPE IS TEXT SIZE IS 32.
DEFINE FIELD RMU$STORAGE_AREA_ID DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$FLAGS DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$TOTAL_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$EXPANDED_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$FRAGMENTED_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$EXPANDED_FRAGMENT_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$TOTAL_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$FRAGMENTED_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$FRAGMENT_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$PAGE_LENGTH DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$MAX_PAGE_NUMBER DATATYPE IS SIGNED LONGWORD.
DEFINE FIELD RMU$FREE_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$OVERHEAD_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$AIP_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$ABM_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$SPAM_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$INDEX_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$BTREE_NODE_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$HASH_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATES_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$OVERFLOW_BYTES DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$LOGICAL_AREA_ID DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$RELATION_ID DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$RECORD_ALLOCATION_SIZE DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$TOTAL_SPACE DATATYPE IS F_FLOATING.
DEFINE RECORD RMU$ANALYZE_AREA.
    RMU$DATE.
    RMU$AREA_NAME.
    RMU$STORAGE_AREA_ID.
    RMU$FLAGS.
    RMU$TOTAL_BYTES.
    RMU$EXPANDED_BYTES.
    RMU$FRAGMENTED_BYTES.
    RMU$EXPANDED_FRAGMENT_BYTES.
    RMU$TOTAL_COUNT.
    RMU$FRAGMENTED_COUNT.
    RMU$FRAGMENT_COUNT.
    RMU$PAGE_LENGTH.
    RMU$MAX_PAGE_NUMBER.
```

(continued on next page)

Example 2–4 (Cont.) Using RMU Analyze to Create an Oracle CDD/Repository Compatible Record Definition File

```
    RMU$FREE_BYTES.
    RMU$OVERHEAD_BYTES.
    RMU$AIP_COUNT.
    RMU$ABM_COUNT.
    RMU$SPAM_COUNT.
    RMU$INDEX_COUNT.
    RMU$BTREE_NODE_BYTES.
    RMU$HASH_BYTES.
    RMU$DUPLICATES_BYTES.
    RMU$OVERFLOW_BYTES.
    RMU$LOGICAL_AREA_ID.
    RMU$RELATION_ID.
    RMU$RECORD_ALLOCATION_SIZE.
    RMU$TOTAL_SPACE.
END RMU$ANALYZE_AREA RECORD.
```

Table 2–2 describes each of the fields in the DB.RRD record.

Table 2–2 Field Descriptions

Field	Definition
RMU$ABM_COUNT	Contains the number of ABM pages in the storage area
RMU$AIP_COUNT	Contains the number of AIP pages in the storage area
RMU$AREA_NAME	Contains the name of the storage area that was analyzed
RMU$AVAILABLE	Contains the amount of initially available space in the index records
RMU$BTREE_NODE_BYTES	Contains the number of bytes for sorted indexes in the storage area
RMU$COUNT	Contains the number of index nodes
RMU$DATA_COUNT	Contains the number of records
RMU$DATE	Contains the date that the Analyze operation was done
RMU$DUPLICATES_BYTES	Contains the number of bytes for duplicate key values for sorted indexes in the storage area
RMU$DUPLICATE_AVAILABLE	Contains the amount of initially available space in the duplicate records

(continued on next page)

Table 2–2 (Cont.) Field Descriptions

Field	Definition
RMU$DUPLICATE_ COUNT	Contains the number of duplicate records
RMU$DUPLICATE_ DATA_COUNT	Contains the number of duplicate records
RMU$DUPLICATE_ KEY_COUNT	Contains the number of duplicate keys
RMU$DUPLICATE_ USED	Contains the amount of available space used in the duplicate records
RMU$EXPANDED_ BYTES	Contains the total size of the stored data in the logical area after decompression
RMU$EXPANDED_ FRAGMENT_BYTES	Contains the number of bytes in the stored fragments after decompression
RMU$FLAGS (for DB.RRD record)	The three possible values in this field have the following meanings: • 3—this value indicates that data compression is enabled for the logical area • 1—this value indicates that data compression is not enabled for the logical area • 0—this value indicates that the record is a storage area record, not a logical area record

(continued on next page)

Table 2–2 (Cont.) Field Descriptions

Field	Definition
RMU$FLAGS (for INDEX.RRD and PLACEMENT.RRD records)	The eight possible values in this field have the following meanings: • 7—Index is hashed and unique. A full report is generated. • 6– Index is hashed and not unique. A full report is generated. • 5– Index is sorted and unique. A full report is generated. • 4—Index is sorted and not unique. A full report is generated. • 3—Index is hashed and unique. A full report is not generated. • 2—Index is hashed and not unique. A full report is not generated. • 1—Index is sorted and unique. A full report is not generated. • 0—Index is sorted and not unique. A full report is not generated.
RMU$FRAGMENTED_ BYTES	Contains the number of bytes in the stored fragments
RMU$FRAGMENTED_ COUNT	Contains the number of records that are fragmented
RMU$FRAGMENT_ COUNT	Contains the number of stored fragments
RMU$FREE_BYTES	Contains the number of free bytes in the storage area
RMU$HASH_BYTES	Contains the number of bytes for hashed indexes in the storage area
RMU$INDEX_COUNT	Contains the number of index records in the storage area
RMU$INDEX_NAME	Contains the name of the index that was analyzed
RMU$KEY_COUNT	Contains the number of keys
RMU$LEVEL	Contains the maximum number of index levels

(continued on next page)

Table 2–2 (Cont.) Field Descriptions

Field	Definition
RMU$LOGICAL_AREA_ID	Contains the logical area id of the logical area that was analyzed
RMU$MAX_KEY_PATH	Contains the largest number of keys touched to access any of the records
RMU$MAX_PAGE_NUMBER	Contains the page number of the last initialized page in the storage area
RMU$MAX_PAGE_PATH	Contains the largest number of pages touched to access any of the records
RMU$MIN_BUF_PATH	Contains the smallest number of buffers touched to access any of the records
RMU$OVERFLOW_BYTES	Contains the number of bytes for hash bucket overflow records in the storage area
RMU$OVERHEAD_BYTES	Contains the number of bytes used for overhead in the storage area
RMU$PAGE_LENGTH	Contains the length in bytes of a database page in the storage area
RMU$RECORD_ALLOCATION_SIZE	Contains the size of a row when the table was initially defined
RMU$RELATION_ID	Contains the record type of the row in the logical area that was analyzed
RMU$RELATION_NAME	Contains the name of the table for which the index is defined
RMU$SPAM_COUNT	Contains the number of SPAM pages in the storage area
RMU$STORAGE_AREA_ID	Contains the area id of the storage area that was analyzed
RMU$TOTAL_BUFFER_PATH	Contains the total number of buffers touched to access all the records
RMU$TOTAL_BYTES	Contains the total size of the data stored in the logical area

(continued on next page)

Table 2–2 (Cont.) Field Descriptions

Field	Definition
RMU$TOTAL_COMPRESSED_IKEY_COUNT	Contains the total number of bytes occupied by the compressed (level 1) index keys
RMU$TOTAL_COUNT	Contains the total number of records stored
RMU$TOTAL_IKEY_COUNT	Contains the total number of bytes that would have been consumed by the index if compression had not been enabled
RMU$TOTAL_KEY_PATH	Contains the total number of keys touched to access all the records
RMU$TOTAL_PAGE_PATH	Contains the total number of pages touched to access all the records
RMU$TOTAL_SPACE	Contains the number of bytes available for storing user data in the logical area (used space + free space + overhead)
RMU$USED	Contains the amount of available space that is used

The RMU Analyze Index command in Example 2–5 outputs the results into an RMS record definition file called INDEX.RRD that is compatible with the repository.

Example 2–5 Using RMU Analyze Index to Create an Oracle CDD/Repository Compatible Record Definition File

```
$ RMU/ANALYZE/INDEX/BINARY_OUTPUT=RECORD_DEFINITION=INDEX.RRD mf_personnel
$!
$! Display the INDEX.RRD file created by the previous command:
$ TYPE INDEX.RRD
DEFINE FIELD RMU$DATE DATATYPE IS DATE.
DEFINE FIELD RMU$INDEX_NAME DATATYPE IS TEXT SIZE IS 32.
DEFINE FIELD RMU$RELATION_NAME DATATYPE IS TEXT SIZE IS 32.
DEFINE FIELD RMU$LEVEL DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$FLAGS DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$USED DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$AVAILABLE DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATE_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATE_USED DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATE_AVAILABLE DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$KEY_COUNT DATATYPE IS F_FLOATING.
```

(continued on next page)

Example 2–5 (Cont.) Using RMU Analyze Index to Create an Oracle CDD/Repository Compatible Record Definition File

```
DEFINE FIELD RMU$DATA_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATE_KEY_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATE_DATA_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$TOTAL_COMPRESSED_IKEY_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$TOTAL_IKEY_COUNT DATATYPE IS F_FLOATING.
DEFINE RECORD RMU$ANALYZE_INDEX.
    RMU$DATE.
    RMU$INDEX_NAME.
    RMU$RELATION_NAME.
    RMU$LEVEL.
    RMU$FLAGS.
    RMU$COUNT.
    RMU$USED.
    RMU$AVAILABLE.
    RMU$DUPLICATE_COUNT.
    RMU$DUPLICATE_USED.
    RMU$DUPLICATE_AVAILABLE.
    RMU$KEY_COUNT.
    RMU$DATA_COUNT.
    RMU$DUPLICATE_KEY_COUNT.
    RMU$DUPLICATE_DATA_COUNT.
    RMU$TOTAL_COMPRESSED_IKEY_COUNT.
    RMU$TOTAL_IKEY_COUNT.
END RMU$ANALYZE_INDEX RECORD.
```

Table 2–2 describes each of the fields in the INDEX.RRD record.

The RMU Analyze Placement command in Example 2–6 outputs the results into an RMS record definition file called PLACEMENT.RRD that is compatible with the repository.

Example 2–6 Using RMU Analyze Placement to Create an Oracle CDD/Repository Compatible Record Definition File

```
$ RMU/ANALYZE/PLACEMENT/BINARY_OUTPUT=RECORD_DEFINITION=PLACEMENT.RRD -
_$ mf_personnel
$!
$! Display the PLACEMENT.RRD file created by the previous command:
$ TYPE PLACEMENT.RRD
DEFINE FIELD RMU$DATE DATATYPE IS DATE.
DEFINE FIELD RMU$INDEX_NAME DATATYPE IS TEXT SIZE IS 32.
DEFINE FIELD RMU$RELATION_NAME DATATYPE IS TEXT SIZE IS 32.
DEFINE FIELD RMU$LEVEL DATATYPE IS SIGNED WORD.
```

(continued on next page)

Example 2–6 (Cont.) Using RMU Analyze Placement to Create an Oracle CDD/Repository Compatible Record Definition File

```
DEFINE FIELD RMU$FLAGS DATATYPE IS SIGNED WORD.
DEFINE FIELD RMU$COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATE_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$KEY_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATE_KEY_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DATA_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$DUPLICATE_DATA_COUNT DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$TOTAL_KEY_PATH DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$TOTAL_PAGE_PATH DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$TOTAL_BUFFER_PATH DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$MAX_KEY_PATH DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$MAX_PAGE_PATH DATATYPE IS F_FLOATING.
DEFINE FIELD RMU$MIN_BUF_PATH DATATYPE IS F_FLOATING.
DEFINE RECORD RMU$ANALYZE_PLACEMENT.
    RMU$DATE.
    RMU$INDEX_NAME.
    RMU$RELATION_NAME.
    RMU$LEVEL.
    RMU$FLAGS.
    RMU$COUNT.
    RMU$DUPLICATE_COUNT.
    RMU$KEY_COUNT.
    RMU$DUPLICATE_KEY_COUNT.
    RMU$DATA_COUNT.
    RMU$DUPLICATE_DATA_COUNT.
    RMU$TOTAL_KEY_PATH.
    RMU$TOTAL_PAGE_PATH.
    RMU$TOTAL_BUFFER_PATH.
    RMU$MAX_KEY_PATH.
    RMU$MAX_PAGE_PATH.
    RMU$MIN_BUF_PATH.
END RMU$ANALYZE_PLACEMENT RECORD.
```

Table 2–2 describes each of the fields in the PLACEMENT.RRD record.

See the *Oracle RMU Reference Manual* for more information on the syntax and use of the [No]Binary_Output qualifier. See the *Oracle Rdb7 Guide to Database Design and Definition* for information on how to use the RMU Load Rms_Record_Def command to load data into an Oracle Rdb database.

2.2 Performance Monitor

Oracle Rdb collects and maintains statistical information about certain database activities as they occur over time. You can use the Performance Monitor to view database statistics and identify various performance problem areas. You can then use the SQL ALTER DATABASE statement to modify the database.

2.2.1 Using Database Statistics

Database statistics are maintained in a global section or shared memory partition on each computer system that runs Oracle Rdb. In a cluster environment, Oracle Rdb maintains statistics separately for each node. In such an environment, the database statistics only reflect activity that originates from the node in the cluster from which the Performance Monitor is invoked.

You invoke the Performance Monitor with the RMU Show Statistics command. Database statistics are collected by the Performance Monitor until:

- The database is closed

  A database is closed when the last process detaches from the database. A Performance Monitor process keeps a database open. Database statistics are reset to zero when the database is closed.

- The time specified with the Until qualifier

  If you specify a time with the Until qualifier when you issue an RMU Show Statistics command, statistics collection ends at the specified time (the database does not have to close first). See the *Oracle RMU Reference Manual* for the complete syntax of the RMU Show Statistics command.

- The Reset option is specified from either the horizontal menu on a screen or with the Reset qualifier on the RMU Show Statistics command line. The Unreset option on the horizontal menu reverses the effect of the Reset option.

  Reset stops the current collection of statistics and starts a collection of new statistics.

If you want to keep the database open for an extended period of time to observe processing activity, you can explicitly open the database using the RMU Open command.

You can disable the writing of database statistics for a process by using the logical name RDM$BIND_STATS_ENABLED or the configuration parameter RDB_BIND_STATS_ENABLED. By default, the writing of database statistics is enabled for each process on a node; the RDM$BIND_STATS_ENABLED or RDB_BIND_STATS_ENABLED value is set to 1 or true for all processes. When you use the RDM$BIND_STATS_ENABLED logical name or the RDB_BIND_STATS_ENABLED configuration parameter to disable database statistics for a process, the Performance Monitor does not collect any statistics for that process; the statistics displayed on the screens will not include information on processes for which statistics have been disabled. Disabling statistics is useful for static, performance-critical applications that have been previously tuned and do not need the information provided by the Performance Monitor.

You disable the writing of database statistics for a process by setting the RDM$BIND_STATS_ENABLED or RDB_BIND_STATS_ENABLED value to 0. The following example shows how to disable database statistics on OpenVMS:

```
$ DEFINE RDM$BIND_STATS_ENABLED 0
```

To enable the writing of database statistics for a process in which the collection of database statistics is disabled, define the logical name or configuration parameter specifying the value 1, or deassign the logical name or configuration parameter. The following example shows how to enable database statistics on OpenVMS:

```
$ ! Set the RDM$BIND_STATS_ENABLED value to 1:
$ DEFINE RDM$BIND_STATS_ENABLED 1
$ !
$ ! Or, deassign the logical name:
$ DEASSIGN RDM$BIND_STATS_ENABLED
```

You can disable the writing of statistics for all processes in one of two ways:

- By setting the RDM$BIND_STATS_ENABLED logical name or RDB_BIND_STATS_ENABLED configuration parameter to 0 for each process.

  OpenVMS VAX OpenVMS Alpha Defining RDM$BIND_STATS_ENABLED as a group or system logical can help simplify this task. ♦

- By disabling statistics collection for the database itself. When statistics collection is disabled for a database, no statistics are displayed for any of the processes attached to the database. You can disable statistics collection for a database by using the STATISTICS COLLECTION IS DISABLED clause of the SQL CREATE DATABASE, ALTER DATABASE, or IMPORT statement. Oracle Corporation recommends that you use SQL syntax to disable and enable statistics collection. The following ALTER DATABASE statement disables statistics collection for the mf_personnel database:

  ```
  SQL> ALTER DATABASE FILENAME mf_personnel
  cont> STATISTICS COLLECTION IS DISABLED;
  ```

  To enable statistics collection for a database, use the STATISTICS COLLECTION IS ENABLED clause of the SQL CREATE DATABASE, ALTER DATABASE, or IMPORT statement. The following ALTER DATABASE statement enables statistics collection for the mf_personnel database:

  ```
  SQL> ALTER DATABASE FILENAME mf_personnel
  cont> STATISTICS COLLECTION IS ENABLED;
  ```

  By default, statistics collection is enabled for a database.

You can display statistical information about:

- Disk I/O operation
- Memory usage
- Locks, including wait times
- Index activity
- Transaction durations
- Snapshot file usage
- Buffer activity
- Row cache activity
- Process information
- Database parameter information
- After-image journaling overhead
- After-image journals on the current node

You can display statistics in the following formats:

- Graphic display
- Numeric display
- Time plots of individual fields (such as transactions per second)
- Scatter plot display
- Tabular display

For more information on the display formats, see Section 2.2.6.1 through Section 2.2.6.5.

2.2.2 Syntax for the RMU Show Statistics Command

Use the RMU Show Statistics command to invoke the character-cell interface of the Performance Monitor. The basic syntax for this command is:

```
RMU Show Statistics [ database-file-name ]
```

Refer to the *Oracle RMU Reference Manual* for a full explanation of the RMU Show Statistics command qualifiers.

2.2.3 Selecting a Display Mode in the Character-Cell Interface

The Performance Monitor operates in three modes: online, record, or replay.

- Online

  In online mode, database activity is monitored as it occurs. By default, Oracle RMU records new statistics every 3 seconds. You can specify a preferred time interval with the Time qualifier when you invoke the Performance Monitor. You can also change the time interval at any time during an interactive session; select the Set_rate choice from the menu and specify the desired interval.

  You can display the statistics on your terminal by using the Interactive qualifier, or you can record the statistics in a binary file by using the Output qualifier, or you can do both.

  Oracle Rdb normally collects statistics for a database as long as the database is open. You can use the Until qualifier to specify a time at which statistics collection will stop. The Until qualifier is typically used with the Output qualifier in a batch job.

- Record

  You can record statistics in a binary file by using the Output qualifier. The binary file does not produce a legible printed listing. You must read the binary file using replay mode; use the RMU Show Statistics command and specify the name of the binary file as an argument to the Input qualifier. You can also write your own program to reformat the file. More information on the binary file can be found in Section 2.2.13.

- Replay

  In replay mode, database activity previously recorded in a binary file is read and displayed on your terminal. The statistics would have been recorded in online mode using the Output qualifier.

  To replay previously recorded statistics, use the Input qualifier and specify the name of the file that contains the statistics. You can then use the arrow keys and the menus to change the display format or the screen, just as in online mode. By using replay mode, you can view a sampling of statistics many times and examine various aspects of the sampling. This is valuable if you find something on one screen, and you want to look at another screen to confirm your suspicions.

  You can change the rate of the display by specifying the desired rate with the Time qualifier or by using the Set_rate menu option. Changing the display rate in replay mode is not the same as in online mode. For example, suppose you gathered statistics into a file in online mode using the default interval of 3 seconds. When you replay statistics from the file, changing the display rate changes the rate at which those 3-second intervals are displayed; it does not change the statistics themselves.

The Select Input Control menu allows you to control the display of database statistics in replay mode. See Section 2.2.4 for more information.

You can use the Nointeractive qualifier to suppress the interactive display. This is useful if you want to generate a binary statistics output file for later reference, but do not want an online display while statistics are being collected.

Several statistics monitor database root file activity. For a multifile database, the .rdb file is the database root file that contains only database header information, not user data.

2.2.4 Using the Select Input Control Menu in the Character-Cell Interface

When the Performance Monitor is in replay mode, type I to select the Input option from the current screen's horizontal menu. When you select the Input option, the Select Input Control menu for replay mode is displayed:

```
        Select Input Control

    A. Replay
    B. Pause
    C. Continue
    D. Place Markpoint
```

The following are the available menu options and their meanings:

- A. Replay

  Causes the input file to be replayed from the beginning. This can be selected from either replay mode or paused mode. When you select the Replay option, statistics are displayed from the beginning of the input file.

- B. Pause

  Causes the current display to pause. When you select the Pause option, the Performance Monitor is in paused mode (note that the "Mode:" field in the display header changes from Replay to Paused). The display remains in paused mode until you select the Continue option from the Select Input Control menu. When you select the Pause option, the Select Input Control menu for paused mode is displayed. In this menu, option B is Single-Step:

  ```
          Select Input Control

      A. Replay
      B. Single-Step
      C. Continue
      D. Place Markpoint
  ```

Two new options (Continue and Single-Step) appear in the horizontal menu for the current screen when pause mode is being used.

All the horizontal menu items for a screen continue to operate normally in paused mode, except Set_rate (you can still change the display rate using the Set_rate option, but the statistics themselves are not changed). In paused mode, for example, you can change the display format from Numbers to Graphic format and back again. The Options and Write options are especially useful, because you know exactly what you are printing. You can also move between the available pages using the right angle bracket (>) and left angle bracket (<) keys when the screen is paused. This allows you to acquire a true snapshot of a moment in time.

To advance by one record in the input file while in paused mode, you can select either option B (Single-Step) from the Select Input Control menu or type S to select the Single-Step option from the horizontal menu. When you are in paused mode, you can also select Replay mode (option A from the Select Input Control menu) and then step through all or part of the records in the input file using the Single-Step option.

- C. Continue

  Causes the screen to resume normal display at the previously selected rate. You can also type C to select the Continue option from the horizontal menu, which returns the Performance Monitor to replay mode. When replay mode resumes, the Continue and Single-Step options are removed from the horizontal menu and the Set_rate option is replaced in the horizontal menu. Also, the Select Input Control menu reverts to displaying only the replay mode options.

- D. Place Markpoint

  Allows you to mark the current record in the input file so you can return to it quickly later. Oracle RMU uses the time the record was created in the input file for the markpoint (this is the time displayed at the top of the display header). When you select this option, a new option is added to the Select Input Control menu:

```
        Select Input Control

    A. Replay
    B. Pause
    C. Continue
    D. Place Markpoint
    E. Goto Markpoint 11:17:59
```

- E. Goto Markpoint <time>

Quickly returns you to the previously saved markpoint, which is identified by the time shown as part of the menu option. When you select option E, you advance or return to the markpoint.

You cannot remove a markpoint, but you can set a new markpoint by selecting D from the Select Input Control menu again. Use markpoints to identify the location of something that is significant to you. Currently, only one markpoint can be established at a time. When you select the Goto Markpoint option, statistics are displayed from the markpoint in the input file.

2.2.5 Navigating in the Performance Monitor

To select the screens that provide many different statistics on database activity, use the menus provided with the Performance Monitor. After you invoke the Performance Monitor, the first screen appears. At the bottom of the first screen is its horizontal menu. For example:

```
--------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
--------------------------------------------------------------------------
```

Each screen has a horizontal menu in which the first letter of each menu selection is highlighted. Make a selection from the menu by typing the highlighted letter of your choice. Type M to view a menu of the available screens:

```
                               Select Display

    A. Summary IO Statistics            O. IO Statistics (by file)  [->
    B. Summary Locking Statistics       P. Locking (one lock type)  [->
    C. Summary Object Statistics        Q. Locking (one stat field) [->
    D. Summary Cache Statistics         R. Lock Statistics (by file)[->
    E. Summary Cache Unmark Statistics  S. Database Parameter Info  [->
    F. Record Statistics                T. Row Cache (One Cache)    [->
    G. Transaction Duration (Total)     U. Row Cache (One Field)    [->
    H. Custom Statistics                V. Row Cache Information    [->
    I. Snapshot Statistics              W. Index Information        [->
    J. Process Information      [->     X. General Information      [->
    K. Journaling Information   [->     Y. Objects (one stat type)  [->
    L. Hot Standby Information  [->     Z. Objects (one stat field) [->
    M. IO Statistics            [->     0. Database Dashboard       [->
    N. Global Buffer Information[->     1. Online Analysis & Info.  [->
```

You can select a screen in the following ways:

- Type the letter of the desired screen.
- Use the arrow keys to move the cursor and highlight the desired screen, and then press the Return key.

You can use the up arrow and down arrow keys to move up and down within the menu and the left arrow and right arrow keys to move between the two columns of the menu.

- Use the plus (+) and minus (–) keys.

  The plus key moves you forward through screens and the minus key moves you backward through screens. When you enter the plus or minus key, the Performance Monitor prompts you to enter the number of screens to move forward or backward respectively. The Performance Monitor moves by the number of screens that you specify. If you enter +0, the Performance Monitor moves to the last screen. If you enter –0, the Performance Monitor moves to the first screen.

- Use the Goto screen options in the tools or help facility.

  See Table 2–4 for a description of these options.

If you are viewing one screen and want to move to another without using the display menu or the tools facility, you can do so by using the left arrow key or Prev Screen key and the right arrow key or Next Screen key. Pressing the left arrow key or Prev Screen key moves you to the previous screen, and pressing the right arrow key or Next Screen key moves you to the next screen. The order of the screens is the order shown on the Select Display menu. This means, for example, when you are viewing the Transaction Duration screen (choice G on the Select Display menu), you can move to the Record Statistics screen (choice F) by pressing the left arrow key once or to the Custom Statistics screen (choice H) by pressing the right arrow key once.

Some displays are longer than one page. The display header indicates the number of pages in a display. You view the succeeding pages of a display by pressing the right angle bracket (>) key, as indicated by the >next_page option in the horizontal menu. Then, to move back to previous pages, press the left angle bracket (<) key, as indicated by the <prev_page option in the horizontal menu. The right angle bracket (>) key stops at the last page of a display and the left angle bracket (<) key stops at the first page.

You can also use the up arrow key and the down arrow key to migrate through displays that contain multiple pages. The arrow keys have the added advantage of migrating in a circular manner. When the first page in a display is the current screen, pressing the up arrow key will move to the last page of the display while pressing the down arrow key at the last page will move to the first page of the display.

When an option from the Select Display menu is followed by [->, this means that after you select the option, the Performance Monitor will display a submenu of options.

For example, if you select the IO Statistics (by file) display, the Select File menu is displayed:

```
        Select File

A. File IO Overview
B. Device IO Overview
C. Device Information
D. root file
E. AIJ file
F. RUJ file
G. ACE file
H. all data/snap files
I. data file MF_PERS_DEFAULT
J. data file EMPIDS_LOW
K. data file EMPIDS_MID
L. data file EMPIDS_OVER
M. data file DEPARTMENTS
N. data file SALARY_HISTORY
O. data file JOBS
P. data file EMP_INFO
Q. <<more>>
```

To select the database file for which you want to display I/O statistics, type the letter of the desired file, or move the cursor to highlight the desired file and then press the Return key. Note that if all of the database files cannot be displayed on a single screen, you can view additional Select File menu choices by choosing the <<more>> option (choice Q) on the first screen of the menu.

From the Select Display, Select File, Select Lock Type, and Select Lock Field menus, pressing the following keys produces the following responses:

- Up arrow

  Moves you up the list of menu choices.

- Down arrow

  Moves you down the list of menu choices.

- Left arrow

  In a two-column menu, moves you to the left column. Has no effect in a one-column menu.

- Right arrow

  In a two-column menu, moves you to the right column. Has no effect in a one-column menu.

- Ctrl/W on OpenVMS and Ctrl/L on Digital UNIX

  Refreshes the screen.

- Ctrl/Z on OpenVMS and Ctrl/D on Digital UNIX

  Cancels the menu.

In a horizontal menu, the first character of each menu selection is highlighted. You make a selection from a horizontal menu by typing the highlighted character of your choice. Table 2–3 shows the menu selections that may be available (depending on the screen) and the responses when the selection is chosen.

Table 2–3 Performance Monitor Horizontal Menu Selections

Alarm	Allows you to specify a duration that a process must stall before it appears on the Stall Messages screen.
Bell	Activates or deactivates the alarm bell.
Brief	Displays the brief version of a screen. Available only for screens with both a brief and full version.
Config	Allows you to configure the statistics screens.
Continue	Causes the screen to resume normal display at the previously selected rate.
Exit	Exits you from the Performance Monitor. Pressing Ctrl/Z on OpenVMS, and Ctrl/D on Digital UNIX from the horizontal menu also exits you from the Performance Monitor.
Filter	Allows you to filter stall messages.
Full	Displays the full version of a screen. Available only for screens with both a brief and full version.
Graph	Displays statistics for the screen in a graphical format.
Help	Invokes help on using the keyboard and on the current screen and its fields.
Input	Displays the Select Input Control menu.
LockID	Displays a submenu of Lock IDs.
Menu	Displays the Select Display menu.
Normal	Puts you back into the normal screen display, showing the screen as it looked before you entered the Time Plot or Scatter Plot display.
Numbers	Displays statistics for the screen in a numbers (chart) format.
Options	Allows you to write screens for a particular time to a file called STATISTICS.RPT in your default directory. You can write the file in graph format, numbers format, or in both formats.

(continued on next page)

Table 2–3 (Cont.) Performance Monitor Horizontal Menu Selections

Pause	Causes the screen to pause the output. Only screens that have the Pause menu option are paused. Other screens continue to be updated and cannot be paused. Press the P key again to release the pause.
Refresh	Refreshes the screen.
Reset	Resets database statistics to zero.
Set_rate	Changes the collection interval to the interval you specify.
Step	Advances by one record in the input file while in paused mode.
Time_plot	Provides a detailed graph of event counts for a particular field.
Unreset	Reverses the effect of the Reset option.
Update	Allows you to change the value of a database attribute.
Write	Writes the contents of any screen, except Help, to a file called RMU.SCR in your default directory.
X_plot	Provides a vertical histogram for a particular field.
Yank	Displays a menu of statistics fields you can select to display custom statistics. The selected fields will automatically appear on the Custom Statistics screen.
Zoom	Displays detailed information about a specific item.
>Next_ page	Pressing the right angle bracket key moves you to the next page of the current screen.
<Prev_ page	Pressing the left angle bracket key moves you to the previous page of the current screen.
!	Invokes the tools facility and displays the Select Tool submenu.
$	Invokes the DCL command facility on OpenVMS.
#	Displays the submenu for the current screen.
+	Advances *n* screens forward.
–	Advances *n* screens backward.

Often, it is necessary to quickly locate a Performance Monitor screen that contains activity, because it is not always apparent what database activity is occurring. You can use the space bar to initiate a search for the next data screen in the current submenu that has current activity. If there is no screen in the current submenu that has activity, the Performance Monitor places you at the next screen, exactly as if you had used the Next Screen key. Also, computational screens (such as the Row Cache Status) and informational screens (such as the Stall Messages, Monitor Log, or AIJ Journal Information screens) are ignored during the search for active data. The search mode is available during replay of a binary input file.

2.2.6 Selecting a Display Format in the Character-Cell Interface

You can display information gathered by the Performance Monitor in the following formats:

- Graph
- Numbers
- Time Plot
- Scatter Plot
- Table

The Graph display format shows the statistics pictorially so you can quickly get an idea of how the database is performing. The Numbers display format shows, in a chart, the actual numbers gathered. The Time Plot display format, available for some screens, allows you to monitor a particular field of the display over time. The Scatter Plot display format, available for some screens, allows you to monitor the current rate of a particular field in vertical histogram form. The Table display format presents statistics in tabular form.

Each screen, regardless of its format, has the same header information:

```
--------------------------------------------------------------------------------
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor  2-MAY-1996 12:40:40
Rate: 3.00 Seconds           Summary IO Statistics          Elapsed: 00:00:29.05
Page: 1 of 1     RDBVMS_USER1:[LOGAN.V7.TMP]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
```

The first line of the header contains the node name, the utility name and version number, and the current system date and time (which is automatically updated at the specified Set_rate interval). The second line contains the screen refresh rate in seconds, the name of the screen, and the elapsed time since the last Reset command. The third line contains the current page number within the screen, the name of the current database, and the Performance Monitor display mode (online, record, or replay).

Section 2.2.6.1 through Section 2.2.6.5 provide more information on display formats.

2.2.6.1 Graphic Display Format

The Graph display format shows occurrence-per-second rates for such events as transactions, verb successes and failures, I/O activity in the database, and other data characteristics of a particular screen. The bars of the graph reflect current event rates; that is, the occurrences-per-second of a given event during the last sample interval. The following is an example of a graphic display:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 17-MAY-1996 14:10:24
Rate: 3.00 Seconds           Summary IO Statistics          Elapsed: 00:02:32.50
Page: 1 of 1          SQL$DISK1:[USER]MF_PERSONNEL.RDB;1            Mode: Online
--------------------------------------------------------------------------------
statistic......... max. cur.         10        20        30        40        50
name.............. rate rate +---------+---------+---------+---------+---------+
                             |         |         |         |         |         |
transactions          1    0 |         |         |         |         |         |
verb successes       80   80 +---------+---------+---------+---------+-------->|
verb failures        11   11 +---------+*        |         |         |         |
                             |         |         |         |         |         |
synch data reads     17   17 +---------+------*  |         |         |         |
synch data writes     0    0 |         |         |         |         |         |
asynch data reads     0    0 |         |         |         |         |         |
asynch data writes    0    0 |         |         |         |         |         |
RUJ file reads        0    0 |         |         |         |         |         |
RUJ file writes       0    0 |         |         |         |         |         |
AIJ file reads        0    0 |         |         |         |         |         |
AIJ file writes       0    0 |         |         |         |         |         |
ACE file reads        0    0 |         |         |         |         |         |
ACE file writes       0    0 |         |         |         |         |         |
root file reads       7    0 |         |         |         |         |         |
root file writes      1    0 |         |         |         |         |         |
--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

The layout of the graphic display is explained in the following list:

- statistic name

  This column identifies the statistic. Table 2–5 lists the screens and indicates where you can find information on each screen.

- max rate

  The maximum occurrence-per-second rate of the event since the Performance Monitor was invoked. You can reset this counter by typing R to select the Reset menu option.

- cur rate

  The occurrence-per-second rate of the event during the last sample interval. You can set the sample interval in advance by using the Time qualifier with the RMU Show Statistics command, or you can change it after initiating the display by typing S to select the Set_rate option.

Bars on the chart reflect the current rate, but are limited by the width of the chart to values of 50. For current rate values less than 50, the bar ends with an asterisk. For current rate values greater than 50, the bar ends with a right angle bracket; the actual value is shown in the current rate column.

You can use the [No]Histogram qualifiers to specify the initial display mode to be used by the Performance Monitor. If you use the Histogram qualifier, statistics will be displayed initially in the Graph display mode. If you use the Nohistogram qualifier (the default), statistics will be displayed initially in the Numbers display mode.

Digital UNIX

The following command on Digital UNIX causes the Performance Monitor to display statistics for the mf_personnel database initially in the Graph display mode:

```
$ rmu -show statistics -histogram mf_personnel
```

♦

When the Performance Monitor is displaying statistics in the Graph display mode, you can switch to the Numbers display mode by typing N.

2.2.6.2 Numbers Display Format

The Numbers display format presents statistics in a columnar chart. The chart provides averages and total count values as well as maximum and current rates. The following is an example of a Numbers display format:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 17-MAY-1996 14:37:03
Rate: 3.00 Seconds          Summary IO Statistics         Elapsed: 05:39:13.29
Page: 1 of 1         SQL$DISK1:[USER]MF_PERSONNEL.RDB;1           Mode: Online
-------------------------------------------------------------------------------
statistic.........      rate.per.second.........     total...     average..
name..............      max.     cur.     avg...     count...     per.trans

transactions              0        0       0.0            2           1.0
verb successes            2        0       1.7          250         125.0
verb failures             0        0       0.2           32          16.0

synch data reads          0        0       0.4           66          33.0
synch data writes         0        0       0.0            0           0.0
asynch data reads         0        0       0.0            0           0.0
asynch data writes        0        0       0.0            0           0.0
RUJ file reads            0        0       0.0            0           0.0
RUJ file writes           0        0       0.0            0           0.0
AIJ file reads            0        0       0.0            0           0.0
AIJ file writes           0        0       0.0            0           0.0
ACE file reads            0        0       0.0            0           0.0
ACE file writes           0        0       0.0            0           0.0
root file reads           0        0       0.1           20          10.0
root file writes          0        0       0.0            5           2.5
-------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

The layout of the Numbers display format is explained in the following list:

- statistic name

  This column identifies the statistic. Table 2–5 lists the screens and indicates where you can find information on each screen.

- rate per second (max)

  The maximum occurrence-per-second rate of the event since the Performance Monitor was invoked. You can reset this counter by typing R to select the Reset option.

- rate per second (cur)

  The occurrence-per-second rate of the event during the last sample interval. You can set the sample interval in advance by using the Time qualifier with the RMU Show Statistics command, or you can change it after initiating a display by typing S to select the Set_rate option.

- rate per second (avg)

  The average occurrence-per-second rate of the event since the database was opened on this node. To reset the counter, type R to select the Reset menu option. After you reset the counter, this column shows the average occurrence-per-second rate of the event since the counter was reset.

- total count

  The total number of occurrences of the event since the database was opened on this node. To reset the counter, type R to select the Reset menu option. After you reset the counter, this column shows the total number of occurrences of the event since the counter was reset.

- average per trans

  The quotient of the total-count column divided by the number of completed transactions.

You can use the [No]Histogram qualifiers to specify the initial display mode to be used by the Performance Monitor. If you use the Histogram qualifier, statistics will be displayed initially in the Graph display mode. If you use the Nohistogram qualifier (the default), statistics will be displayed initially in the Numbers display mode.

Digital UNIX

The following command on Digital UNIX causes the Performance Monitor to display statistics for the mf_personnel database initially in the Numbers display mode:

```
$ rmu -show statistics -nohistogram mf_personnel
```

♦

When the Performance Monitor is displaying statistics in the Numbers display mode, you can switch to the Graph display mode by typing G.

2.2.6.3 Time Plot Display Format

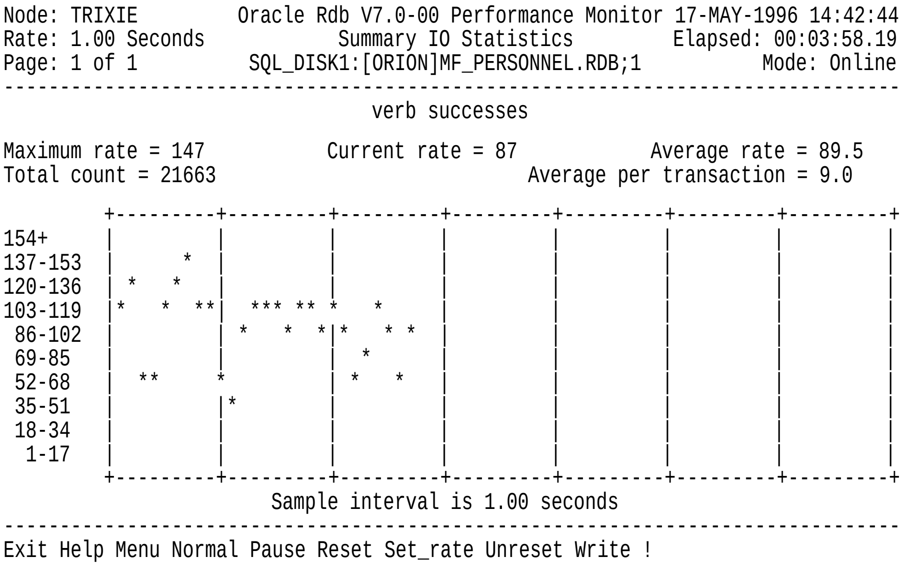
The Time Plot display format provides a detailed graph of event counts for a particular field on the screen. Each screen includes a horizontal menu. If the menu lists Time_plot, you can type T to request a time plot. After you type T, select a field from the screen by typing the letter associated with the desired field (or by using the arrow keys to move the cursor to the desired field and then pressing the Return key).

The following display is an example of a time plot that shows verb success rates over time:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 17-MAY-1996 14:41:24
Rate: 3.00 Seconds            Summary IO Statistics         Elapsed: 00:07:35.33
Page: 1 of 1           SQL$DISK1:[USER]MF_PERSONNEL.RDB;1           Mode: Online
--------------------------------------------------------------------------------
                                 verb successes

Maximum rate = 147           Current rate = 144           Average rate = 78.0
Total count = 10216                            Average per transaction = 8.9

         +---------+---------+---------+---------+---------+---------+---------+---------+
100+     |       ^^^^^^^^**^^^^        |         |         |         |         |         |
 89-99   |      *  |         |         |         |         |         |         |         |
 78-88   | **      |         |         |         |         |         |         |         |
 67-77   |    **   |         |         |         |         |         |         |         |
 56-66   |*        |         |         |         |         |         |         |         |
 45-55   |   *     |         |         |         |         |         |         |         |
 34-44   |         |         |         |         |         |         |         |         |
 23-33   |         |         |         |         |         |         |         |         |
 12-22   |         |         |         |         |         |         |         |         |
  1-11   |         |         |         |         |         |         |         |         |
         +---------+---------+---------+---------+---------+---------+---------+---------+
                        Sample interval is 3.00 seconds
--------------------------------------------------------------------------------
Exit Help Menu Normal Pause Reset Set_rate Unreset Write !
```

The circumflex characters (^) in the top row indicate that the number of events (verb successes in this case) is larger than the highest range value of the chart. To change the range of the display, select the Reset menu option.

If you change the sample interval using the Set_rate menu option, the display is rescaled to the new interval. For example, the following Summary I/O Statistics display uses a sample interval of 1 second:



```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 17-MAY-1996 14:42:44
Rate: 1.00 Seconds            Summary IO Statistics         Elapsed: 00:03:58.19
Page: 1 of 1          SQL_DISK1:[ORION]MF_PERSONNEL.RDB;1          Mode: Online
--------------------------------------------------------------------------------
                                 verb successes

Maximum rate = 147           Current rate = 87            Average rate = 89.5
Total count = 21663                            Average per transaction = 9.0

         +---------+---------+---------+---------+---------+---------+---------+
154+     |         |         |         |         |         |         |         |
137-153  |      *  |         |         |         |         |         |         |
120-136  | *   *   |         |         |         |         |         |         |
103-119  |*   *  **|  *** ** *   *     |         |         |         |         |
 86-102  |         | *   *  *|*   * *  |         |         |         |         |
 69-85   |         |         |  *      |         |         |         |         |
 52-68   |  **     *         | *   *   |         |         |         |         |
 35-51   |         |*        |         |         |         |         |         |
 18-34   |         |         |         |         |         |         |         |
  1-17   |         |         |         |         |         |         |         |
         +---------+---------+---------+---------+---------+---------+---------+
                        Sample interval is 1.00 seconds
--------------------------------------------------------------------------------
Exit Help Menu Normal Pause Reset Set_rate Unreset Write !
```

The notepad facility provides an option that allows you to enable or disable the Time_plot baseline. Enabling the baseline updates the graph continuously. You can enable the Time_plot baseline by performing the following steps:

1. Enter the exclamation point (!) to invoke the tools facility.
2. Select the Notepad options from the Select Tool submenu.
3. Select Enable Time_plot option from the Select Tool Option submenu.

2.2.6.4 Scatter Plot Display Format

The Scatter Plot display format, similar to the Time Plot display format, shows information for a selected statistics field on the current screen. However, the information in the scatter plot is displayed in a vertical histogram. The Scatter Plot display format allows you to determine the composition of the values that determine the average rate statistic. If the horizontal menu lists X_plot, you can type X to request a scatter plot. After you type X, select a field from the screen by typing the letter associated with the desired field (or by using the arrow keys to move the cursor to the desired field and then pressing the Return key).

After you have selected a field, the Scatter Plot display initially shows that the scale of the display is based on the running average rate for the selected field. This scale can be changed using the Reset menu option.

Each vertical line of the screen identifies a collection rate bucket based on the selected statistic rate. The complete set of buckets is known as the collection range and comprises the entire Scatter Plot screen. For example, the Scatter Plot screen may have a range from 0.0 to 1.5 seconds, with each bucket representing 0.1 (one-tenth) of a second.

Based on the selected screen refresh rate, the normalized current rate for the selected statistic field is examined to determine into which bucket the rate would occur. This algorithm results in a current rate scattered distribution graph, which is extremely beneficial for performance analysis.

The following example shows a Scatter Plot display that is interesting because the current rate distribution is significantly more than the running average (indicated by the letter R on the bottom horizontal axis). This indicates that some event has recently occurred that resulted in sudden increased asynchronous data read activity. Also, notice the skewed distribution of the data.

```
Node: ORANOD         Oracle Rdb V7.0-00 Performance Monitor 29-MAY-1995 10:13:33
Rate: 0.50 Seconds           Summary IO Statistics        Elapsed: 2 19:36:03.15
Page: 1 of 1         CADD$:[J_DOE.WORK.ALS]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------
Out-of-range count:         0.1--- = 0             ❶ 3.0+++ = 29
Current statistic range:       low = 0.8           ❷   high = 4.0
                               synch data writes   ❸
Scaled distribution of statistic rate (in seconds)❹      Elapsed: 00:25:00.51
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+---+
|    |    |    |    |    |    |    |    ***  |    **** |    |    |    |    |   |
|    |    |    |    |    |    |    |    ******    *****|    |    |    |    |   |
|    |    |    |    |    |    |    |    ***************|    |    |    |    |   |
|    |    |    |    |    |    |    |    ******************  |    |    |    |   |
|    |    |    |    |    |    |    |    ********************|    |    |    |   |
|    |    |    |    |    |    |    |    ********************|    |    |    |   |
|    |    |    |    |    |    |    |    *********************    |    |    |   |
|    |    |    |    |    |    |    |    *************************|    |    |   |
|    |    |    |    |    |    |    |    ***************************** |    |   |
|    |    |    |    *    *    ***  **** ********************************** *** |
+----+----+----+----+----+-R--+----+----+----+----A----+----+--M-+----+----+---+
                           ❺                    ❻          ❼
0.0  0.2  0.4  0.6  0.8  1.0  1.2  1.4  1.6  1.8  2.0  2.2  2.4  2.6  2.8  3.0+
                  (Each "*" represents 11 statistics updates)
--------------------------------------------------------------------------------
Config Exit Help Menu Normal Pause Reset Set_rate Write !
```

The example shows the following types of information about the summary region of the display:

❶ Identifies the number of statistics updates that are below the minimum recorded rate (XXX- - -) or above the maximum recorded rate (XXX+++). This is information about the boundary data conditions.

❷ Identifies the lowest and highest recorded rate, which are used as the basis for rescaling the display when you select the Reset on-screen menu option. This is also information about the boundary data conditions.

❸ Region identifies the statistic whose rate distribution information is being collected.

❹ The fourth line of the summary region identifies the scaling factor of the display, and the total elapsed time of the collection.

Three sliding indicators are displayed along the bottom horizontal axis:

❺ The letter R indicates the running average rate, which is computed over the total elapsed time as shown in the summary region. Note that the R running average rate may not always appear, depending on the collection range. Also, resetting the elapsed time is useful for comparing the running average against the collected average.

❻ The letter A indicates the collected average rate, which is computed on the collection time.

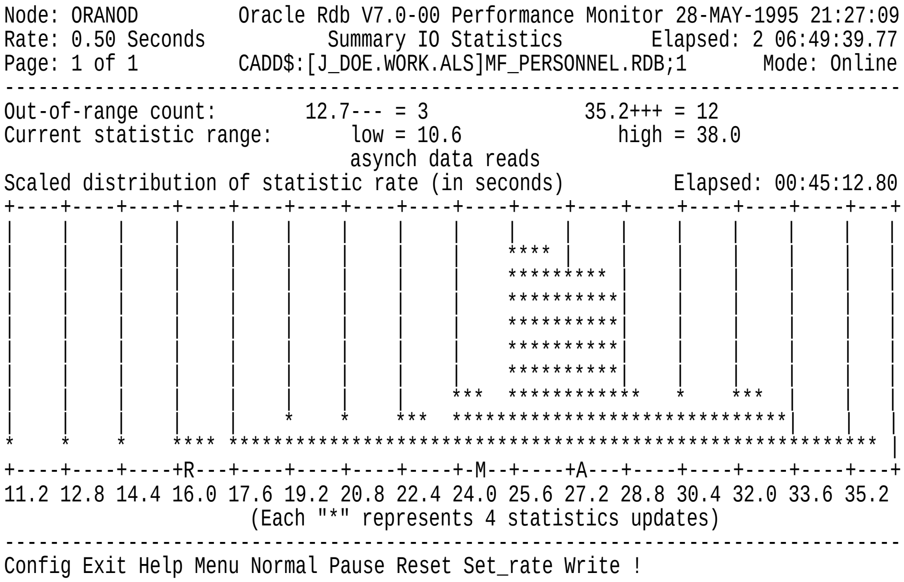
❼ The letter M indicates the collected median rate, which is also computed on the collection time.

The number of collection buckets is based on the display width of the terminal. By default, each bucket comprises 5 columns of the display; the number of columns can be configured by the user. For example, setting the terminal width to 132 columns allows you to display more information than with an 80-column terminal width. Decreasing the number of columns per bucket achieves the same effect on an 80-column terminal width, but with a corresponding loss of precision.

Notice that the collected median is significantly higher than the collected average. This is indicative of skewed data, typically caused by periodic blips in the run-time performance. Identifying and eliminating these blips produces a smoother-running system.

Notice also that these occasional blips tend to get lost as noise in the running average. However, removing these blips ultimately improves the overall running average of the system.

Ideally, the Scatter Plot display should resemble the classical bell-shaped curve data distribution. The following example shows a display with this behavior:



```
Node: ORANOD        Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1995 21:27:09
Rate: 0.50 Seconds          Summary IO Statistics       Elapsed: 2 06:49:39.77
Page: 1 of 1        CADD$:[J_DOE.WORK.ALS]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Out-of-range count:       12.7--- = 3            35.2+++ = 12
Current statistic range:      low = 10.6            high = 38.0
                              asynch data reads
Scaled distribution of statistic rate (in seconds)         Elapsed: 00:45:12.80
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+---+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
|    |    |    |    |    |    |    |    |    **** |    |    |    |    |    |   |
|    |    |    |    |    |    |    |    |    ********* |    |    |    |    |   |
|    |    |    |    |    |    |    |    |    **********|    |    |    |    |   |
|    |    |    |    |    |    |    |    |    **********|    |    |    |    |   |
|    |    |    |    |    |    |    |    |    **********|    |    |    |    |   |
|    |    |    |    |    |    |    |    |    **********|    |    |    |    |   |
|    |    |    |    |    |    |    |    ***  ************   *    ***  |    |   |
|    |    |    |    |    *    *    ***  ******************************|    |   |
*    *    *    **** ********************************************************** |
+----+----+----+R---+----+----+----+----+-M--+----+A---+----+----+----+----+---+
11.2 12.8 14.4 16.0 17.6 19.2 20.8 22.4 24.0 25.6 27.2 28.8 30.4 32.0 33.6 35.2
                      (Each "*" represents 4 statistics updates)
--------------------------------------------------------------------------------
Config Exit Help Menu Normal Pause Reset Set_rate Write !
```

The Scatter Plot screen includes the following menu options:

- Normal

  Puts you back into the normal screen display, showing the screen as it looked before you entered the Scatter Plot display.

- Reset

  Instead of resetting the statistics values, resets the Scatter Plot scale based on the lowest and highest rate values actually recorded. This menu option is very useful for focusing the display on the actually occurring statistics update rates.

- Set_rate

  Causes the Scatter Plot information to be cleared and data collection restarted. This is required because the new rate's data collection patterns are unrelated to the old rate's data collection patterns.

- Config

  Config allows you to fine-tune the rate collection parameters and the display appearance. Currently, you can perform two configuration operations: set the number of displayed columns per bucket, and set the rate collection range. Both of these settings allow you more control over what data is collected and displayed.

Decreasing the number of columns per bucket allows you to collect more rates. Conversely, increasing the number of columns per bucket allows you to collect fewer rates, but with more accurate rate reporting (that is, better scaling) per bucket. The minimum number of columns is 2 and the maximum is 10.

Setting specific collection range values allows you to determine the actual collection range, which is normally based on the rates already collected.

The Scatter Plot screen information is recorded in the binary output file created using the Output qualifier, and can be replayed using the Input qualifier. Note that changes in collection rates during the recording phase are not identified during replay and may cause interesting Scatter Plot displays.

2.2.6.5 Table Display Format

Some screens are available only in the Table display format. The Stall Messages, Active User Stall Messages, and Defined Logicals displays, for example, are in Table display format.

The following is an example of the Defined Logicals display:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 17-MAY-1996 14:44:03
Page: 1 of 3                   Defined Logicals           Elapsed: 05:46:46.72
Rate: 3.00 seconds    SQL$DISK1:[USER]MF_PERSONNEL.RDB;1           Mode: Online
--------------------------------------------------------------------------------
Logical.Name................ Table.Name......... Logical.Definition..........
RDM$BIND_BUFFERS             LNM$SYSTEM_TABLE    80
SQL$DATABASE                 LNM$PROCESS_TABLE   SQL_PERSONNEL
RDM$MAILBOX_CHANNEL          LNM$SYSTEM_TABLE    MBA2563:
RDM$MONITOR                  LNM$SYSTEM_TABLE    SYS$SYSROOT:[SYSEXE]






------------------------------------------------------------------------------
Exit Full Help Menu >next_page <prev_page Set_rate Write !
```

2.2.7 Using the Zoom Menu Option in the Character-Cell Interface

A **zoom screen** is a subwindow that displays detailed information about a specific screen item, typically a lock, process, AIJ journal, or storage area. Note that a zoom screen is a static snapshot of information. Unlike normal statistics screens, the information does not change. A screen that has zoom capability displays the zoom option in its horizontal menu.

It is often necessary to quickly review detailed information about a storage area or active database process. Typically, this need occurs while you are reviewing one of the by-area or per-process screens and some event occurs that warrants further investigation.

For example, the following Checkpoint Information screen is interesting because one of the processes has not checkpointed:

```
Node: ORANOD         Oracle Rdb V7.0-00 Performance Monitor 21-APR-1996 06:30:30
Rate: 0.10 Seconds           Checkpoint Information        Elapsed: 00:28:45.14
Page: 1 of 1       DD$:[ANDERS.WORK.ALS]MF_PERSONNEL.RDB;12       Mode: Online
-------------------------------------------------------------------------------
Process.ID  Ckpt.Vno:Ckpt.Vbn  QuietVno
2020D41F:1s       32      106         0
2020EC26:1s                           0
20208957:1        32      132         0
20207EF7:1        32      119         0
202084F9:1        32      135         0

-------------------------------------------------------------------------------
Config Exit Help Menu >next_page <prev_page Pause Set_rate Write Zoom !
```

Selecting the Zoom menu option provides information on why process 2020EC26 has not checkpointed. After pressing the Z key, a menu of the currently displayed process IDs is presented; select the desired process ID. The Performance Monitor then displays detailed information about the specified process. A zoom screen similar to the following is displayed:

```
Node: ORANOD         Oracle Rdb V7.0-00 Performance Monitor 21-APR-1996 06:32:11
Rate: 0.10 Seconds           Checkpoint Information        Elapsed: 00:30:26.38
Page: 1 of 1     DD$:[ANDERS.WORK.ALS]MF_PERSONNEL.RDB;12          Mode: Online
-------------------------------------------------------------------------------
Process.ID  Ckpt.Vno:Ckpt.Vbn  QuietVno
2020D41F:1s       34       89         0
2020EC26:1s                           0
20208957:1        34      164         0
20207EF7:1        34      169         0
202084F9:1        34      163         0

+--Process Information: 2020EC26---------------------------------+
|                                                                 |
| Active user with process ID 2020EC26 (database server)         |
| User name is RDBVMS                                             |
| Process name is RDM_LCS701_1                                    |
| Image name is DSA1:[SYS1.SYSCOMMON.][SYSEXE]RDMLCS701.EXE;206   |
| No transaction in progress                                      |
| Monitor ID is 1                                                 |
| Internal Stream ID is 1                                         |
| Internal Transaction ID is 655                                  |
|                                                                 |
+-----------------------------------------------------------------+

-------------------------------------------------------------------------------
Exit Help Menu >Next_page <Prev_page Pause Set_rate Write Zoom !
```

As you can see, process 2020EC26 is a database server process and does not have a transaction in progress.

2.2.8 Writing a Display to a File in the Character-Cell Interface

When you select Options from the main menu of the Performance Monitor, you can write most of the screens for a particular time to a file called STATISTICS.RPT. You can write the file in either graph or numbers format, or both graph and numbers format.

When you select Options from the horizontal menu, Oracle Rdb displays the following:

```
+---Select Report Format---+
|                          |
|  Write Report (Graph)    |
|  Write Report (Numbers)  |
|  Write Report (Both)     |
|                          |
+--------------------------+
```

Your choices are as follows:

- Write Report (Graph)

  Writes all the Graph screens to a file called STATISTICS.RPT when you issue the command.

- Write Report (Numbers)

  Writes all the Numbers screens to a file called STATISTICS.RPT when you issue the command.

- Write Report (Both)

  Writes both the Graph and Numbers screens to a file called STATISTICS.RPT when you issue the command.

You can also write the contents of any screen, except Help, when you select the Write option. This option creates a file, called RMU.SCR, in your default directory.

2.2.9 Using Performance Monitor Tools in the Character-Cell Interface

The Performance Monitor provides a set of tools to create and maintain a notepad and to invoke several predefined and any user-defined commands. The commands can be invoked while executing the Performance Monitor.

You can invoke the tools facility by entering the exclamation point (!) character on any screen or menu. The Select Tool submenu is displayed; for example:

```
+------ Select Tool --------+
|                           |
|  A. Goto screen "by-name" |
|  B. Goto screen       [-> |
|  C. Notepad (EDT)         |
|  D. Edit a file (EDT)     |
|  E. Send/Receive Mail     |
|  F. Invoke command        |
|  G. Wake up process       |
|  H. Terminate image       |
|  I. Terminate process     |
|  J. Set process priority  |
|  K. Switch database       |
|  L. Start screen cycling  |
|  M. Start stall logging   |
|  N. Start DBKEY logging   |
|  O. Disable auto-refresh  |
|  P. Start ALS process     |
|  Q. <<more>>              |
+---------------------------+
```

Selecting Option Q displays the following:

```
+------ Select Tool --------+
|                           |
|  A. <<more>>              |
|  B. Wake up RCS process   |
|  C. Invoke ABS process    |
|  D. Suspend ABS process   |
|  E. Notepad options   [-> |
+---------------------------+
```

The Select Tool submenu supports the options in Table 2–4.

Table 2–4 Select Tool Options

Option	Description
Goto screen "by name"	Allows you to enter the case-insensitive name of the screen you wish to go to, or any portion of the screen name. Note that certain screen names are not unique unless you have entered a sizable portion of the screen name. If you enter a response that is not unique, the first matching screen will be selected. Note that the selections are not necessarily in alphabetical order. If you enter a response that does not match any available screen, the Performance Monitor displays an error and you will be left on the current screen.

(continued on next page)

Table 2–4 (Cont.) Select Tool Options

Option	Description
Goto screen [->	Displays a multipage menu of all available screen names, in alphabetical order. The selectable names of the screens depend on the screens available to the Performance Monitor. For example, the global buffer screen names are not selectable unless they are actually available.
Notepad (EDT)[1]	The notepad is a file that you can use to maintain notes about a session. The name of the notepad file can be specified using the Notepad options submenu. The default notepad file name is STATS.NOTEPAD. You can maintain the notepad file by using one of three text editors: EDT, LSE, or TPU. You can change the editor by using the Options submenu (described later) or by defining the RMU$EDIT logical name. The value for the RMU$EDIT logical name must be EDT, LSE, or TPU. The default text editor is EDT. If you have an EDTINI.EDT file, it is read when the text editor is invoked. If the notepad file does not exist, it is automatically created containing four lines of information before the designated text editor is invoked. The information identifies the product version, the database name, the notepad file name, as well as the date and time the notepad file was originally created. You might find it useful to define the notepad file name to be one of the other Performance Monitor file names, such as RMU.SCR, which is the name of the file used for writing screens. You can now create an image of a particular screen using the Write menu option, and then document it using the notepad facility.
Edit a file (EDT)[1]	Invokes a text editor. You are prompted with Filename: to enter an optional file name. If you do not want to specify a file name, press the Return key. You can invoke one of three text editors: EDT, LSE, or TPU. You can change the editor by using the Options submenu (described later) or by defining the RMU$EDIT logical name. The value for the RMU$EDIT logical name must be EDT, LSE, or TPU. The default editor is EDT. If you have an EDTINI.EDT file, it is read when the text editor is invoked.
Send/Receive Mail[1]	Invokes the mail utility.

[1] Supported on OpenVMS systems only

(continued on next page)

Table 2–4 (Cont.) Select Tool Options

Option	Description
Invoke command	Invokes a user-defined command. On OpenVMS, the DCL ($) prompt is then displayed at the bottom of the current screen and you can enter any DCL command. On OpenVMS, you can also invoke the DCL command facility by entering the dollar sign ($) on any screen. The $ prompt is then displayed on the bottom of the current screen, and you can enter any DCL command. On Digital UNIX, the right angle bracket (>) is displayed at the bottom of the current screen and you can enter any Digital UNIX command. You should be careful when invoking any utility (such as SQL) that operates against the database. Any command that requires exclusive database access will hang indefinitely.
Wake up process[2]	Wakes up a database process that is hibernating. This option should only be necessary following catastrophic database failure where the database recovery process (DBR) was unable to wake up the process. This option has no effect on a process that is not hibernating; however, take care not to overuse this option. This option is limited to operations performed on specific database processes; you cannot use this option on arbitrary processes on the system.
Terminate image[2]	Terminates a database image with the exception "%RDMS-F-IMGABORTED, image aborted at privileged user request." Note that the process itself is not terminated, only the image is terminated. This is the preferred operation as this operation does not cause the database to be frozen during process recovery. This option is limited to operations performed on specific database processes; you cannot use this option on arbitrary processes on the system.
Terminate process[2]	Terminates a database process without warning. This operation causes the database to be frozen while the database recovery process (DBR) recovers the terminated process. This option is limited to operations performed on specific database processes; you cannot use this option on arbitrary processes on the system.

[2] On OpenVMS, you need the Group privilege to operate on another process in the same group, unless the process has the same user identification code (UIC) as the invoking process. You need the World privilege to operate on another process in the system.

(continued on next page)

Table 2–4 (Cont.) Select Tool Options

Option	Description
Set process priority[1]	Allows you to change the priority for the specified process on OpenVMS. You need the ALTPRI (alter priority) privilege to set the priority higher than the base priority of the specified process.
Switch database	Allows you to switch databases without having to exit from the Performance Monitor. You are prompted for the file specification for the new database you want to open. You can use command recall to display previous terminal inputs. You can use Ctrl/Z on OpenVMS and Ctrl/D on Digital UNIX to cancel the request. When you enter a new database file specification, the current database is closed and the new database is opened. If the new database cannot open for any reason, the previously opened database is opened again. The ability to open a new database is not available if you are replaying a binary input file, using the Input qualifier, or recording a binary output file, using the Output qualifier.
Start screen cycling	Directs the Performance Monitor to continually cycle through the set of screens associated with the currently selected screen. Each screen is displayed for the number of seconds specified. When the Performance Monitor is cycling through screens, you can change screen modes or change submenus; cycling through the screens associated with your choice will continue at whichever menu level is currently selected. When you select the "Start screen cycling" option, you are prompted to enter a time interval (in seconds). The specified value for the time interval must be greater than or equal to the value specified for the Time qualifier. In addition, if you manually change the refresh rate (using the Set_rate menu option) to a value that is greater than the value you specify for the time interval, the cycling is performed at the interval you specify with the Set_rate menu option.

[1] Supported on OpenVMS systems only

(continued on next page)

Table 2–4 (Cont.) Select Tool Options

Option	Description
Start stall logging	Specifies that stall messages are to be written to the stall log file. You are prompted to enter a file name for the stall log. Writing stall messages to a log file can be useful when you notice a great number of stall messages being generated, but do not have the resources to immediately investigate and resolve the problem. The log file generated can be reviewed later so that the problems can be traced and resolved. The stall messages are written to the file in a format similar to the Stall Messages screen. Stall messages are written to the file at the same rate as the screen refresh rate. Specifying a large refresh rate minimizes the size of the file, but results in a large number of missed stall messages. Specifying a small refresh rate produces a large log file, but contains more of the stall messages generated. You do not need to display the Stall Messages screen to record the stall messages to the log file. The stall log is maintained regardless of which screen, if any, is displayed. By default, stall messages are not logged to a file.
Start DBKEY logging	Logs the records accessed during a given processing period by the various attached processes. You are prompted to enter a file name where all the accessed dbkeys are logged. The dbkey information is written at the current screen refresh rate. Using a larger refresh rate minimizes the size of the file but results in a large number of missed dbkey messages. Using a smaller refresh rate produces a large log file, but contains a much finer granularity of dbkey messages. Note that you do not need to display the DBKEY Information screen to record the dbkey messages to the dbkey log. The dbkey log is maintained regardless of which screen, if any, is displayed.
Disable auto-refresh	Broadcast messages overwrite the horizontal menu in the Performance Monitor. By default, the horizontal menu is refreshed after the message is displayed. This option allows you to disable refreshing of the screen.
Start ALS process	Starts or stops the AIJ log server process if AIJ journaling is enabled.
Wake up RCS process	Instructs the row cache server process to initiate a sweep even when a threshold has not been reached.

(continued on next page)

Table 2–4 (Cont.) Select Tool Options

Option	Description
Invoke ABS process	Invokes the AIJ backup server (ABS) process for a specific noncurrent AIJ journal.
Suspend ABS process	Temporarily suspends or resumes the ABS process.
Notepad options [->[3]	Changes various tools options. The Select Tool Option submenu is displayed. Option A selects the EDT text editor. This is the default editor. Option B selects the LSE text editor, if it exists. If the LSE editor does not exist, then the EDT editor is selected. Option C selects the TPU text editor, if it exists. If the TPU editor does not exist, then the EDT editor is selected. Option D changes the notepad file name. The default file name for the notepad is STATS.NOTEPAD. Option E enables or disables the Time_plot baseline. Enabling the baseline updates the Time_plot display graph continuously. The Select Tool Option submenu remains displayed until you enter Ctrl/Z on OpenVMS or Ctrl/D on Digital UNIX.

[3] Enable Time_plot baseline is the only notepad option supported on Digital UNIX.

2.2.10 Getting Online Help in the Character-Cell Interface

An extensive online help facility is available to help you understand and interpret the statistical displays. You select the Help menu for the Performance Monitor by pressing the H key from a horizontal menu, or by pressing the PF2 key or Help key. The following is an example of the Performance Monitor help menu:

```
A. Using the keyboard
B. Screen description
C. Field descriptions    [->
D. Search for help on... [->
E. Goto screen "by-name"
F. Goto screen           [->
```

The Help menu displays the available Help topics. When you select “Using the keyboard,” the help text explains menu choices and other keys. When you select “Screen description,” Help displays text that describes the current screen. When you select “Field descriptions,” press the letter associated with the field you want help on or use the arrow keys to position the cursor on the field. Then, when you press the Return key, Help displays an explanation of the field. The “field descriptions” choice does not appear on the Help menu if a display does not provide help text for fields.

In some cases, the help text for keyboard, screen, and field descriptions is more than one page in length. You can scroll through multiple help screens, either a line at a time or a page at a time. Use the up arrow key to move the help text up one line at a time and the down arrow key to move the help text down one line at a time. Use the Prev Screen or left arrow key to scroll the help text backward one page at a time, and the Next Screen or right arrow key to scroll the help text forward one page at a time.

When one or more lines of help text is available in either the forward or reverse direction, the arrow keys and the Next Screen and Prev Screen keys are activated. Instructions at the bottom of the screen indicate whether these keys are activated. If you press the left arrow, right arrow, Prev Screen, or Next Screen keys when they are not activated, nothing happens. Also, the help screen displays the indicator “MORE–>” while one or more lines of help exist in the forward direction and “<–MORE” while one or more lines of help exist in the reverse direction.

The name of the help topic that you are reading about appears at the top of the help screen.

The size of the help text is based on the number of lines configured for the terminal; for instance, a DECwindows terminal with 48 lines has a larger viewport than a VT100 terminal with 24 lines.

2.2.11 Types of Database Statistics

The statistics that you can see when you use the Performance Monitor fall into the following categories:

- Statistics that provide general information about what is happening with your database, such as transaction access patterns and the load on a system.
- Statistics that are tied to parameters whose settings you can modify to improve database performance, such as the PIO statistic where the buffer pool overflows and a marked page must be written back to the database.
- Statistics that display dynamic information that automatically changes to reflect database parameter modifications, such as buffer information and row cache information.
- Statistics that allow you to do some online analysis of your database. For example, the Transaction Analysis screen examines the ratio of verb successes to verb failures.

The way you interpret these statistics depends on how your database is structured and how it is used. If you are confident that your database is well-designed, you should use the steady-state approach to analyzing the statistics. In the steady-state approach, you monitor the statistics regularly until you become familiar with the normal performance of the database. Then you note any deviations from the norm and adjust either the database design or certain system parameters (depending on the problems you encounter) to resolve the problem. If you are not confident that your database is well-designed and you are experiencing performance problems, you can use the statistics to try to pinpoint the problem areas.

Quite often, an application is written when a database is relatively small. While the database is small, performance might be acceptable. It is often only when the database grows that performance degrades. In this case, you can gather statistics on the database as it grows and compare the differences in transaction rates. The statistics will show you how the database has changed over time. Comparing the statistics should help you highlight the cause of a problem. By comparing statistics from different database sizes, you can identify the level at which degradation began.

Regardless of how you track database activity, important statistics to monitor include the following:

- On the I/O screen:

  Compare the verb successes statistic to the verb failures statistic. A high failure rate compared to the success rate indicates that many things are failing. The problem might be too many deadlocks, but is really application-dependent.

  Compare the .ruj file reads and the .ruj file writes statistics. Too many .ruj file reads indicate too many rollbacks. Too many .ruj file writes might indicate contention problems or transactions that modify a lot of data. More than one write for each transaction indicates that marked pages are being written back to the database before the actual commit (meaning either another recovery-unit requested the page, the buffer pool overflowed, or there are very large transactions with lots of updates).

- On the summary lock screen:

  Look at the rqsts not queued, rqsts stalled, and rqst deadlocked statistics. High numbers for these statistics (relative to the norm for your database) indicate contention problems. The problem might be that your database design is not optimal. The problem could also be a high number of blocking asynchronous system traps (ASTs) or it may be that users are contending for frequently accessed data.

- On the AIJ screen:

  Compare the records written statistic to the blocks written statistic.

  The ratio of records written to blocks written gives an indication of the size of the AIJ records (or the size of the record modified). This statistic is only informational.

- On the PIO screen:

  Compare the data page request statistic to the SPAM page request statistic. If the SPAM fetch rate is high relative to the data fetch rate, it might simply be an accurate reflection of the data access patterns. However, because it might also be an indication of contention problems, you should try to determine the cause of a high number of SPAM fetches.

  Look at the pool overflow statistic. If there are many pool overflows for each transaction, you might need to adjust the buffer size or the number of buffers allocated for each user. You may also want to consider enabling global buffers. Refer to Section 4.1.2 for information on managing buffers.

  Look at the unmark buffer statistic and check the pool overflow and lock conflict statistics. If the buffer had to be unmarked due to a pool overflow or a lock conflict, Oracle Rdb is doing extra work. This may be the result of lock contention or a buffer pool being too small.

  Compare the data file reads statistic to the data file writes statistic. Generally, this ratio gives you information about the current transaction mix.

- On the snapshot screen:

  Look at the retrieved record, the fetched line, and the read snap page statistics. If the fetched line number is much higher than that for retrieved record, read-only transactions are doing a lot of work to access their records. The problem might be that the access paths are not optimal.

  The read snap page also indicates the amount of database activity. If the read snap page number is higher than the number for retrieved record, the records fetched may be those that have been frequently updated. A zero indicates that read-only transactions are rarely accessing data modified by update transactions.

  Check the page too full statistic, which applies to read/write transactions. This statistic indicates that some data pages contain frequently accessed data that form chains of snapshot pages.

Look at the page in use or page conflict statistics. These statistics indicate contention within the snapshot file. The problem might be that the snapshot file should be extended. The file might not extend itself because it can resolve the contention by doing extra work. However, you might want to extend the snapshot file manually to see if it helps alleviate the problem.

2.2.12 Understanding the Performance Monitor Screens

To move from one screen to another, select the Menu option to see the main menu. Then select the desired screen.

Although only one screen can be active at any one time, all the statistics counters (including maximum and averages) are maintained. If you specify the Output qualifier, you can select different screens without affecting the format of the binary statistics output file. The file contains all the database statistics, not just the current screen.

Table 2–5 provides a brief description of each screen and indicates where you can find more information about each one.

Table 2–5 Performance Monitor Screens

Screen	Description	Reference
Summary IO Statistics	Shows a summary of database I/O activity.	Section 3.2.1.1
Summary Locking Statistics	Shows a summary of the locking activity.	Section 3.8.1.3 Section 8.4
Summary Object Statistics	Shows cumulative information for all database root file objects.	Performance Monitor help
Summary Cache Statistics	Shows cumulative information for all row caches in the database.	Performance Monitor help
Summary Cache Unmark Statistics	Shows cumulative row cache “unmark” statistics that describe how rows in the row caches are written back to disk.	Performance Monitor help
Record Statistics	Shows a summary of data row activity.	Section 4.1.1.7
Transaction Duration	Shows overall real-time transaction processing performance.	Section 3.2.1.6
Custom Statistics	Shows a customized display of any of the base statistics information (for example, non-by-area and non-per-process statistics).	Section 2.2.14

(continued on next page)

Table 2–5 (Cont.) Performance Monitor Screens

Screen	Description	Reference
Snapshot Statistics	Shows snapshot activity for update and read-only transactions.	Section 4.1.1.10
Stall Messages	Shows a summary of database users' stall activity.	Section 3.2.1.3
Active User Stall Messages	Shows all processes attached to the database on the current node and, if a process stalled, shows the reason why the stall occurred.	Section 3.2.1.5
Process Accounting	Shows continuously updated OpenVMS accounting information about local processes in a VMScluster environment.	Section 4.1.1.6
Checkpoint Information	Shows process checkpoint information.	Section 4.1.1.12
Active User Chart	Shows the number of active users attached to the database over a period of time.	Performance Monitor help
CPU Utilization	Shows the current CPU utilization of each database process.	Section 3.3
DBR Activity	Shows one line of information for each database recovery (DBR) process active on the node.	Section 6.5.2
Monitor Log	Allows you to view the monitor log on line.	Performance Monitor help
Defined Logicals	Shows a list of all the logical names currently accessible to the Performance Monitor.	Section 2.3.1
Lock Timeout History	Identifies the object that causes a timeout event.	Section 3.8.1.5
Lock Deadlock History	Identifies the object that causes a deadlock event.	Section 3.8.1.4
DBKEY Information	Shows the last retrieved DBKEY for data, snapshot, SPAM, AIP, and ABM pages.	Section 3.2.1.4
AIJ Statistics	Shows a summary of after-image journaling activity.	Section 4.1.1.8

(continued on next page)

Table 2–5 (Cont.) Performance Monitor Screens

Screen	Description	Reference
Group Commit Statistics	Provides information about AIJ group commit processing	Performance Monitor help
AIJ Journal Information	Shows information about all of a database's after-image journals on the current node.	Section 4.1.1.9
AIJ Journal Growth Trend	Graphically portrays the size of the current AIJ journal over a measured period of time.	Performance Monitor help
ALS Statistics	Shows AIJ log server activity.	Performance Monitor help
2PC Statistics	Provides information about distributed transaction performance.	Performance Monitor help
RUJ Statistics	Provides summary information for all active update transactions.	Performance Monitor help
Checkpoint Statistics	Shows transaction and checkpoint activity.	Section 4.1.1.11
Recovery Statistics	Identifies various recovery phases and shows information on how long each phase took to complete.	Performance Monitor help
Hot Standby Statistics	Provides information about the performance and status of the hot standby feature.	Performance Monitor help
Synchronization Mode Statistics	Provides a breakdown of each type of synchronization mode.	Performance Monitor help
PIO Statistics–Data Fetches	Shows statistics on how data page requests are handled.	Section 4.1.1.3 Section 8.1.3
PIO Statistics–SPAM Fetches	Shows statistics on how SPAM page requests are handled.	Section 4.1.1.4 Section 8.1.3
PIO Statistics–Data Writes	Shows a summary of data file writes and buffer unmarking activity.	Section 4.1.1.2
PIO Statistics–SPAM Writes	Shows SPAM write I/O information.	Performance Monitor help
PIO Statistics–File Access	Shows information about file access.	Performance Monitor help
Asynchronous IO Statistics	Shows statistics on asynchronous reads and writes to the database files.	Section 4.1.1.5

(continued on next page)

Table 2–5 (Cont.) Performance Monitor Screens

Screen	Description	Reference
IO Stall Time	Shows a summary of all I/O stall activity.	Section 3.2.1.2
GB Utilization	Shows the utilization of each page in a global buffer.	Performance Monitor help
GB Hot Page Information	Displays a list of the most heavily shared pages in the global buffer pool.	Performance Monitor help
GB Frequency Information	Displays the effectiveness of the global buffer pool for sharing of pages.	Performance Monitor help
IO Statistics (by file)	I/O statistics for each database file.	Section 4.2.1.4 Section 8.1.2.3
Device Information	Shows an online view of the storage area device information.	Performance Monitor help
Locking (one lock type)	Shows lock statistics for one particular lock type.	Section 3.8.1.3
Locking (one stat field)	Shows lock statistics for all lock types of a particular statistical field.	Section 3.8.1.3
Lock Statistics (by file)	Provides information about page locks that are specific to storage areas and snapshot files.	Performance Monitor help
Database Parameter Information	Shows dynamic information that automatically changes to reflect database parameter modifications.	Performance Monitor help
Row Cache (One Cache)	Provides summary information for a specific row cache.	Performance Monitor help
Row Cache (One Field)	Shows row cache statistics for all row caches of a particular statistical field.	Performance Monitor help
Row Cache Utilization	Provides utilization information for each row in a specific row cache.	Performance Monitor help
Hot Row Information	Displays a list of the most frequently accessed rows for a specific row cache.	Performance Monitor help
Row Cache Status	Provides overall status for a specific row cache.	Performance Monitor help
Row Cache Queue Length	Provides information to determine the relative CPU performance impact of row caching.	Performance Monitor help

(continued on next page)

Table 2–5 (Cont.) Performance Monitor Screens

Screen	Description	Reference
Row Length Distribution	Shows the distribution of the various row lengths in a specific row cache.	Performance Monitor help
RCS Statistics	Provides information about the run-time operation of the row cache server (RCS) process.	Performance Monitor help
Index Statistics (Retrieval)	Shows sorted index retrieval activity.	Section 3.9.5.2 Section 8.1.3.3
Index Statistics (Insertion)	Shows sorted index modification activity.	Section 3.9.5.2
Index Statistics (Removal)	Shows sorted index deletion activity.	Section 3.9.5.2
Hash Index Statistics	Shows hashed index update and retrieval activity.	Section 3.9.5.2
VM Usage Statistics	Shows dynamic virtual memory usage for all database users on a node.	Section 4.4.1
Name translation	Shows statistics on database dashboard updates and logical name translation.	Performance Monitor help
Objects (one stat type)	Shows statistics for a specific database object.	Section 3.4
Objects (one stat field)	Shows statistics for a specific collection category.	Section 3.4
Database Dashboard	Shows the actual database parameter and attribute settings used by the processes attached to the database.	Section 2.2.15
Online Analysis & Info.	Displays analysis information	Section 2.2.16

2.2.13 Getting Statistics Output in a Formatted Binary Output File in the Character-Cell Interface

Specify the Output qualifier on the RMU Show Statistics command to get statistics output in a formatted binary output file with a .dat default file type.

To replay a formatted binary output file, use the RMU Show Statistics command with the Input qualifier. This file must have been created by an earlier RMU Show Statistics session that specified the Output qualifier. The replay mode was discussed in Section 2.2.3.

If you use the Input qualifier, you can change the rate of replay display by using the Time qualifier. Thus, you can record a session with Time=60 (gather statistics once a minute), and then replay it with Time=1 (update the display once a second). This replay rate will display 10 hours of statistics in 10 minutes. All rates per second in the display are computed based on the original (recording) times, so the replay rate does not affect the display numbers in any way.

OpenVMS VAX OpenVMS Alpha

The following command create a formatted binary output file named database.stats on OpenVMS:

```
$ RMU/SHOW STATISTICS/OUTPUT=database.stats mf_personnel.rdb
```

♦

Digital UNIX

The following command create a formatted binary output file named database.stats on Digital UNIX:

```
$ rmu -show statistics -output=database.stats mf_personnel.rdb
```

♦

The binary output file contains four different record types:

- Header record (4096 bytes, fixed)
- Storage-Area information record (512-byte structures uniformly packed in a variable-size record)

  Each storage area record contains several (one or more) storage area entries, tightly packed. The binary output file header record contains information that describes the unpacking algorithm.
- Base statistics record (4096 bytes, fixed)

  Each base statistics record contains one base statistic entry. No unpacking is necessary.
- By-Area statistics record (512-byte structures uniformly packed in a variable-size record)

  Each by-area statistics record contains several (one or more) by-area statistics entries, tightly packed. The binary output file header record contains information that describes the unpacking algorithm.

The binary output file contains the following format:

```
+--------+---------+--------+---------+--------+---------+
| Header | StArea  |  Base  | By-Area |  Base  | By-Area |
| record | records | record | records | record | records |...
+--------+---------+--------+---------+--------+---------+
         (if by-area)      (if by-area)      (if by-area)
                   ^
                   | start of statistics records
```

The different record types do not have tags identifying them. Each record is placed in a specific sequence, depending on flags and values stored in the header record.

OpenVMS VAX OpenVMS Alpha

The following command on OpenVMS allows you to use the replay mode of the Performance Monitor to view the statistics in the database.stats formatted binary output file:

```
$ RMU/SHOW STATISTICS/INPUT=database.stats
```

♦

OpenVMS VAX OpenVMS Alpha

The following command on Digital UNIX allows you to use the replay mode of the Performance Monitor to view the statistics in the database.stats formatted binary output file:

```
$ rmu -show statistics -input=database.stats
```

♦

OpenVMS VAX OpenVMS Alpha

However, some users want to format the output differently. To make this possible, the format of the binary file is documented in the following text file:

```
SYS$LIBRARY:RMU$SHOW_STATISTICS.CDO
```

The text file RMU$SHOW_STATISTICS.CDO lists and defines all the records available in the Performance Monitor binary output file (if you have installed multiple versions of Oracle Rdb on your system, the text file is named RMU$SHOW_STATISTICSvv.CDO, where vv is the Oracle Rdb software version). The following example shows how the binary output file can be defined in the data repository using the SYS$LIBRARY:RMU$SHOW_STATISTICS.CDO file. Oracle Corporation recommends that you place the definitions contained in this file in its own repository. In the example, the repository is called CDD$TOP.RDB$EXAMPLES.

First, create the new repository:

```
$ REPOSITORY OPERATOR
CDO> DEFINE DIRECTORY CDD$TOP.RDB$EXAMPLES.
```

Next, define the fields and records:

```
CDO> SET DEFAULT CDD$TOP.RDB$EXAMPLES
CDO> @SYS$LIBRARY:RMU$SHOW_STATISTICS.CDO
```

New statistics are created for new versions of Oracle Rdb. The SYS$LIBRARY:RMU$SHOW_STATISTICS.CDO file is also updated with each new version of Oracle Rdb. Existing procedures that use Oracle RMU binary statistics will continue to work with new versions of Oracle Rdb. However, if you want to be able to access the new statistics available with new versions of Oracle Rdb, you should update your binary file definitions and programs with each new version of Oracle Rdb. ♦

2.2.14 Customizing the Performance Monitor Display in the Character-Cell Interface

The Performance Monitor allows you to create a customized statistics screen. Any of the base statistics information (for example, non-by-area and non-per-process) can be customized.

The size of your Custom Statistics screen determines the number of statistics fields you can add. The header and horizontal menu take up eight lines of the Custom Statistics screen; you can add a statistics field on each of the remaining lines, up to a maximum of 36 different fields. Each statistics field is placed on a separate line of the screen.

The Custom Statistics screen is an ordinary screen; information can be displayed graphically as a histogram or in numeric tabular format. Also, the Time_plot and X_plot options are available for any of the selected statistics fields.

To select the Custom Statistics screen, select the Menu option and then move the cursor with the down arrow key to the Custom Statistics option and press Return.

Initially, the Custom Statistics screen contains two statistics: database binds and transactions. These statistics can be moved, removed, or replaced.

There are two different methods available to populate the Custom Statistics screen: fields can either be yanked from an existing statistics screen and implicitly put on the Custom Statistics screen, or you can manually create a statistics field. Oracle Corporation recommends using the yank-and-put method, which is easier.

To yank and put a statistics field, use the Yank menu option on the statistics screen containing the statistics field you want to select. The Yank menu option is not available in the Custom Statistics screen.

When you select Y, a menu of the selectable fields is shown; this menu is similar to the Help on fields menu. Select the letter of the statistics field you want to yank and that field will automatically appear on the Custom Statistics screen. Note that once you select a particular statistics field, you cannot select it again unless you first delete it from the Custom Statistics screen. This means that the field selection menu changes every time you select a field.

The selected field is positioned on the first available row of the Custom Statistics screen. You cannot specify the row of the Custom Statistics screen on which the field will be placed using the yank-and-put method. However, once the field has been put on the Custom Statistics screen, you can change the position of that field.

The Yank menu option is automatically canceled when you have selected all the available statistics fields on any given screen or when no more fields are available on the Custom Statistics screen.

Using the yank-and-put method is the recommended approach for creating the Custom Statistics screen. You browse through the existing screens and select the set of statistics fields you want to monitor.

If you are an advanced user, you can also manually create a statistics field. On the Custom Statistics screen, you can use the Config menu option to explicitly enter the index of the specific statistics field, numbered from 1 to 1020. These indexes are documented in the text file SYS$LIBRARY:RMU$SHOW_STATISTICS.CDO (if you have installed multiple versions of Oracle Rdb on your system, the text file is named SYS$LIBRARY:RMU$SHOW_STATISTICSvv.CDO, where vv is the Oracle Rdb software version). When you select statistics files manually, the keyboard will beep if you specify the index of an existing statistics field. However, no error message is displayed.

When you type C on the Custom Statistics screen, the configuration menu is displayed. The configuration menu allows you to add, delete, move, and compress statistics fields on the Custom Statistics screen. The configuration menu varies according to the state of the Custom Statistics screen but typically all four options are displayed.

When you select the Add menu option, you are prompted to select the screen position where you want to place the statistics field, the index of the statistics field, and the title of the statistics field. The title you specify is then appended to the respective index, in brackets, to show which statistics have been manually selected.

This option is extremely useful for tracking statistics that are not normally displayed. For instance, the statistics field that tracks the total number of bytes of virtual memory (VM) allocated is not displayed by any Performance Monitor screen. However, you can select index number 16 and the number of bytes of VM allocated is displayed on the Custom Statistics screen.

When you select the Delete menu option, you are prompted to select the statistics field you want to delete.

When you select the Move menu option, you are prompted to select the statistics field to be moved and the screen position where the selected field will be displayed.

Selecting the Compress menu option eliminates all intervening blanks lines on the screen. This option is particularly useful after you have deleted one or more statistics fields.

Initially, the Custom Statistics screen appears as follows:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor  3-MAY-1996 09:25:01
Rate: 3.00 Seconds             Custom Statistics            Elapsed: 00:00:21.23
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V7]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name.............       max..... cur..... avg....... count....... per.trans....
database binds                 0        0       0.0             1           0.0
transactions                   0        0       0.0             0           0.0











--------------------------------------------------------------------------------
Config Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write
```

The initial database binds and transactions statistics are displayed.

To copy one or more fields from the VM Usage Statistics screen to the Custom Statistics screen, proceed to the VM Usage Statistics screen:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor  3-MAY-1996 09:49:15
Rate: 3.00 Seconds            VM Usage Statistics           Elapsed: 00:24:35.29
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V7]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------



statistic.........      rate.per.second............. total....... average......
name.............       max..... cur..... avg....... count....... per.trans....

GET_VM calls                 187        0       0.2           351           0.0
FREE_VM calls                  0        0       0.0            14           0.0

GET_VM kilobytes             795        0       1.6          2402           0.0
FREE_VM kilobytes             32        0       0.7           978           0.0

$EXPREG calls                  0        0       0.0             0           0.0



--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

Type Y to display the Yank-and-Put menu:

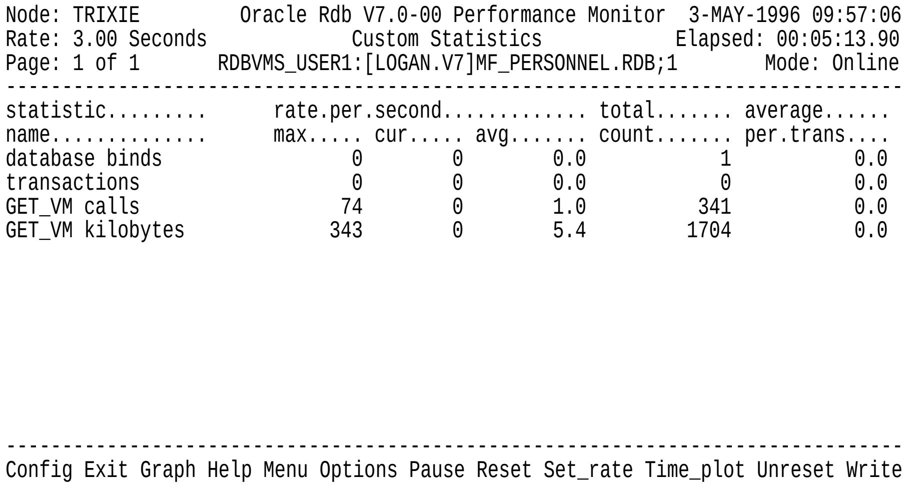
```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor  3-MAY-1996 09:51:56
Rate: 3.00 Seconds           VM Usage Statistics          Elapsed: 00:00:04.65
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V7]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------



statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

A. GET_VM calls             74       0       72.9         339           0.0
B. FREE_VM calls             0       0        0.4           2           0.0

C. GET_VM kilobytes        343       0      336.3        1564           0.0
D. FREE_VM kilobytes        31       0       30.1         140           0.0

E. $EXPREG calls             0       0        0.0           0           0.0



--------------------------------------------------------------------------------
Type <return> or <letter> to select stats field, <control-Z> to cancel
```

Type A to select the GET_VM calls statistics field. You remain in the selection mode but the selection menu changes with each selection. After you select option A, the new menu is displayed as follows:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor  3-MAY-1996 09:51:56
Rate: 3.00 Seconds           VM Usage Statistics          Elapsed: 00:00:04.65
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V7]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------



statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

GET_VM calls                74       0       72.9         339           0.0
A. FREE_VM calls             0       0        0.4           2           0.0

B. GET_VM kilobytes        343       0      336.3        1564           0.0
C. FREE_VM kilobytes        31       0       30.1         140           0.0

D. $EXPREG calls             0       0        0.0           0           0.0



--------------------------------------------------------------------------------
Type <return> or <letter> to select stats field, <control-Z> to cancel
```

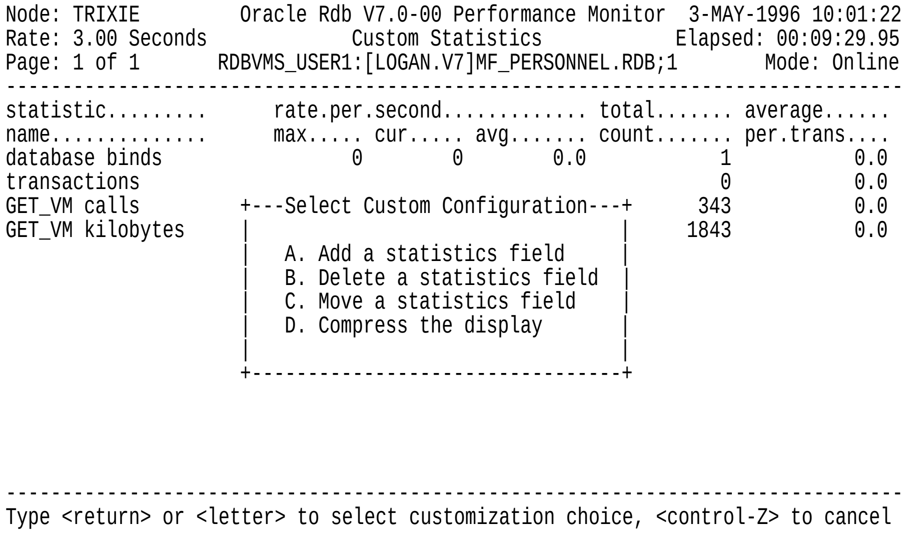
Because you have already selected the GET_VM calls statistics field, you cannot select it again, so it has been removed from the selection menu. To select the GET_VM kilobytes field, type B. Press Ctrl/Z to exit the VM Usage Statistics menu.

Now, return to the Custom Statistics screen. The GET_VM calls and GET_VM kilobytes fields that you selected with the yank-and-put method are displayed:



```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor  3-MAY-1996 09:57:06
Rate: 3.00 Seconds             Custom Statistics            Elapsed: 00:05:13.90
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V7]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....
database binds                 0        0        0.0            1           0.0
transactions                   0        0        0.0            0           0.0
GET_VM calls                  74        0        1.0          341           0.0
GET_VM kilobytes             343        0        5.4         1704           0.0








--------------------------------------------------------------------------------
Config Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write
```

If you do not want to display these statistics fields, type C to choose the Config menu option. This displays the configuration menu:



```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor  3-MAY-1996 10:01:22
Rate: 3.00 Seconds             Custom Statistics            Elapsed: 00:09:29.95
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V7]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....
database binds                 0        0        0.0            1           0.0
transactions                                                    0           0.0
GET_VM calls         +---Select Custom Configuration---+      343           0.0
GET_VM kilobytes     |                                 |     1843           0.0
                     |   A. Add a statistics field     |
                     |   B. Delete a statistics field  |
                     |   C. Move a statistics field    |
                     |   D. Compress the display       |
                     |                                 |
                     +---------------------------------+




--------------------------------------------------------------------------------
Type <return> or <letter> to select customization choice, <control-Z> to cancel
```

To delete an existing statistics field, choose menu option B. The list of existing statistics fields is displayed and you can select the field to be deleted.

Once the configuration operation is complete, the Custom Statistics screen is displayed again:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor  3-MAY-1996 10:21:51
Rate: 3.00 Seconds            Custom Statistics           Elapsed: 00:29:59.74
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V7]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------
statistic.........     rate.per.second............. total....... average......
name..............     max..... cur..... avg....... count....... per.trans....
database binds               0        0       0.0           1          0.0
transactions                 0        0       0.0           0          0.0

GET_VM kilobytes           343        0       1.5        2682          0.0







--------------------------------------------------------------------------------
Config Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write
```

Note the row where the deleted statistics field appeared is now blank. The display is not automatically compressed to eliminate empty lines. Press C from the horizontal menu to select the Config menu, then select D from the Custom Configuration menu to compress (eliminate) the empty lines.

2.2.15 Performance Monitor Database Dashboard Facility

The Performance Monitor provides a Database Dashboard facility for both the global database and individual database processes. You access the facility by selecting the Database Dashboard option from the main menu of the Performance Monitor. The following submenu is displayed:

```
+---------- Select Display -----------+
|                                     |
|  A. IO Dashboard                    |
|  B. Locking Dashboard               |
|  C. AIJ Dashboard                   |
|  D. Checkpoint Dashboard            |
|  E. Hot Standby Dashboard           |
|  F. Row Cache Dashboard             |
|  G. RUJ Dashboard                   |
|  H. Monitor Dashboard               |
|  I. ABS Dashboard                   |
|  J. ALS Dashboard                   |
|  K. DBR Dashboard                   |
|  L. RCS Dashboard                   |
|  M. Per-Process I/O Dashboard       |
|  N. Per-Process Journal Dashboard   |
|  O. Per-Process Row Cache Dashboard |
|                                     |
+-------------------------------------+
```

The Row Cache Dashboards are displayed only if row caching is enabled for your database.

The Database Dashboard facility displays the actual database parameter and attribute settings being used by the processes attached to the database. It provides a way for the database administrator (DBA) to examine logical name and configuration parameter values and other database attributes settings at run time.

OpenVMS VAX ≡ OpenVMS Alpha ≡

Optionally, users are allowed to dynamically update certain database parameters and attributes on a single node at run time. The net effect of these changes can be examined at run time without having to restart database processes. These updates are nonpersistent.

The Database Dashboard facility allows you to “drive” the database faster or slower, and immediately see the impact of increasing or decreasing certain database settings.

You can use the Update menu option, by typing the letter U, to change the value of a database attribute. Before you can update database attributes, you need to start your Performance Monitor session with the Options=Update qualifier, and you need OpenVMS WORLD, BYPASS, and SYSNAM privileges.

Caution

You should use the Update option carefully. Oracle Rdb does not perform error checking on the updated values.

Note that updates made to any attributes are not stored in the database root file. The purpose of updating attributes is to test and measure the effects of changes on the database, so that you can later make persistent changes to appropriate database attributes using interactive SQL.

Database attributes are updated on the current node only. ♦

See the Performance Monitor help for descriptions of the Database Dashboard screens and fields.

2.2.16 Performance Monitor Online Analysis Facility

The Performance Monitor provides an Online Analysis facility that identifies items of interest to a DBA for further investigation into possible performance problems. You access the facility by selecting the Online Analysis & Info. option from the main menu of the Performance Monitor. The following submenu is displayed:

```
+----- Select Display -----+
|                          |
|  A. Buffer Analysis      |
|  B. Transaction Analysis |
|  C. AIJ Analysis         |
|  D. RUJ Analysis         |
|  E. Recovery Analysis    |
|  F. Record Analysis      |
|  G. Area Analysis        |
|  H. Locking Analysis     |
|  I. Index Analysis       |
|  J. Row Cache Analysis   |
|                          |
+--------------------------+
```

Note that the items identified in the Online Analysis screens are not necessarily indications of performance problems. The items may be an indication of a performance problem, but in most cases, further research is necessary.

The Online Analysis screens use the actual database parameters and attributes specified using interactive SQL. The screens do not use the Database Dashboard facility because the Database Dashboard attributes are temporary settings. Therefore, online changes made using the Database Dashboard are not reflected in the analysis output. Yet, using the Database Dashboard facility is the recommended method for testing potential database attribute settings. You must use interactive SQL to make the attribute settings persistent.

The Online Analysis facility is invoked at the designated screen refresh interval. As the Online Analysis facility is fairly CPU-intensive, and the analysis results seldom vary greatly over a minuscule period of time, Oracle Corporation recommends that the screen refresh interval be set to 3 seconds or more.

The Online Analysis screens are not recorded in the binary output file produced using the Output qualifier. Consequently, these screens are not available when you replay a binary file using the Input qualifier.

See the Performance Monitor help for descriptions of the Online Analysis screens.

2.3 Oracle Rdb Logical Names and Configuration Parameters

This section describes how Oracle Rdb logical names and configuration parameters can be used to enhance database performance. Table 2–6 lists the logical names and configuration parameters and provides a brief description of how they can be used in tuning your database. Appendix A describes each logical name and configuration parameter in more detail and illustrates its use.

Table 2–6 Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDB$CHARACTER_SET Not applicable on Digital UNIX	Defines an alternate character set for use by Oracle Rdb.
RDB$LIBRARY RDB_LIBRARY	Specifies a protected library that you can use to store external routine images, such as external functions.
RDB$RDBSHR_EVENT_FLAGS Not applicable on Digital UNIX	Can be used to override the four event flag numbers that are assigned to RDB$SHARE at startup time by the LIB$GET_EF system service.
RDB$REMOTE_BUFFER_SIZE SQL_NETWORK_BUFFER_SIZE	Changes the default buffer size, up to your system quota limits, of network transfers.
RDB$REMOTE_MULTIPLEX_OFF SQL_NETWORK_NUMBER_ ATTACHES	Controls the number of remote server processes used for multiple remote database acesses to the same node.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDB$ROUTINES RDB_ROUTINES	Specifies the location of an external routine image.
RDBVMS$CREATE_DB RDB_CREATE_DB	Determines which users are allowed to create databases using the SQL CREATE DATABASE statement.
RDM$BIND_ABS_LOG_FILE RDB_BIND_ABS_LOG_FILE	Defines a file name for the after-image journal backup server (ABS) log file.
RDM$BIND_ABS_OVERWRITE_ALLOWED RDB_BIND_ABS_OVERWRITE_ALLOWED	Specifies whether the after-image journal backup server (ABS) resets overwritten AIJ journals.
RDM$BIND_ABS_OVERWRITE_IMMEDIATE RDB_BIND_ABS_OVERWRITE_IMMEDIATE	Specifies whether journals are immediately reset. RDM$BIND_ABS_OVERWRITE_ALLOWED or RDB_BIND_ABS_OVERWRITE_ALLOWED must be enabled.
RDM$BIND_ABS_QUIET_POINT RDB_BIND_ABS_QUIET_POINT	Indicates whether the after-image journal backup server (ABS) performs a quiet-point journal backup.
RDM$BIND_ABW_ENABLED RDB_BIND_ABW_ENABLED	Enables or disables asynchronous batch-write operations.
RDM$BIND_AIJ_CHECK_CONTROL_RECS RDB_BIND_AIJ_CHECK_CONTROL_RECS	Specifies whether Oracle Rdb checks for control records during AIJ cache formatting.
RDM$BIND_AIJ_EMERGENCY_DIR RDB_BIND_AIJ_EMERGENCY_DIR	Specifies the location of the emergency AIJ journal.
RDM$BIND_AIJ_IO_MAX RDB_BIND_AIJ_IO_MAX	Allows you to override the default value for the maximum AIJ group commit I/O buffer size. The default value is 127 blocks.
RDM$BIND_AIJ_IO_MIN RDB_BIND_AIJ_IO_MIN	Allows you to override the minimum AIJ group commit I/O buffer size. The default value is 8 blocks.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDM$BIND_AIJ_STALL RDB_BIND_AIJ_STALL	Defines the amount of time, in milliseconds, that a transaction waits after submitting its commit record to the .aij log file.
RDM$BIND_AIJ_SWITCH_GLOBAL_CKPT RDB_BIND_AIJ_SWITCH_GLOBAL_CKPT	Specifies whether Oracle Rdb performs a global checkpoint after an AIJ switch-over has occurred.
RDM$BIND_ALS_CREATE_AIJ RDB_BIND_ALS_CREATE_AIJ	Indicates whether the ALS server is to create an emergency AIJ journal if the AIJ switch-over operation enters the suspended state.
RDM$BIND_APF_DEPTH RDB_BIND_APF_DEPTH	Specifies the number (depth) of buffers for Oracle Rdb to asynchronously prefetch for a process.
RDM$BIND_APF_ENABLED RDB_BIND_APF_ENABLED	Enables or disables asynchronous prefetch operations.
RDM$BIND_BATCH_MAX RDB_BIND_BATCH_MAX	Defines the number of live data page cache buffers that are written to the database as part of a batch-write or asynchronous batch-write operation. By setting this logical name or configuration parameter to a value smaller than the number of cache buffers allocated to the user, you can control the size of the I/O bursts that occur with batch-write operations. This results in more predictable and uniform I/O behavior for the system.
RDM$BIND_BUFFERS RDB_BIND_BUFFERS	Allows you to provide an alternative number of buffers at run time. This can be useful when you need to temporarily override the default number of buffers for a specific task, but in general want to use the default.
RDM$BIND_BUFOBJ_ENABLED Not applicable on Digital UNIX	Specifies whether the OpenVMS Alpha buffer object feature is enabled that locks Oracle Rdb local buffers into physical memory.
RDM$BIND_CBL_ENABLED RDB_BIND_CBL_ENABLED	Specifies whether coarse buffer locking is enabled.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDM$BIND_CKPT_BLOCKS RDB_BIND_CKPT_BLOCKS	Indicates the number of AIJ blocks after which a checkpoint will occur.
RDM$BIND_CKPT_TIME RDB_BIND_CKPT_TIME	Indicates the amount of time, in seconds, after which a checkpoint will occur.
RDM$BIND_CKPT_TRANS_INTERVAL RDB_BIND_CKPT_TRANS_INTERVAL	Defines a process-specific checkpoint interval value based on the number of committed transactions. Fast commit processing must be enabled.
RDM$BIND_CLEAN_BUF_CNT RDB_BIND_CLEAN_BUF_CNT	Specifies the number of clean buffers to be maintained at the end of a process' least recently used queue of buffers for replacement. Specifying an appropriate value with this logical name or configuration parameter can reduce the number of I/O stalls experienced by a process.
RDM$BIND_COMMIT_STALL RDB_BIND_COMMIT_STALL	Defines the amount of time, in milliseconds, that a transaction waits after attempting to become the group commit process.
RDM$BIND_DAPF_DEPTH_BUF_CNT RDB_BIND_DAPF_DEPTH_BUF_CNT	Specifies the number of buffers to prefetch from the physical area. The default is half the number of buffers defined for the database user.
RDM$BIND_DAPF_ENABLED RDB_BIND_DAPF_ENABLED	Enables and disables detected asynchronous prefetch operations.
RDM$BIND_DAPF_START_BUF_CNT RDB_BIND_DAPF_START_BUF_CNT	Specifies the number of buffers to be accessed sequentially from the physical area before detected asynchronous prefetch (DAPF) read operations start.
RDM$BIND_HRL_ENABLED RDM_BIND_HRL_ENABLED	Specifies whether hold retrieval locks are enabled.
RDM$BIND_LOCK_TIMEOUT_INTERVAL RDB_BIND_LOCK_TIMEOUT_INTERVAL	Allows you to set the amount of time a transaction waits for locks to be released.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDM$BIND_MAX_DBR_COUNT RDB_BIND_MAX_DBR_COUNT	Defines the maximum number of database recovery (DBR) processes to be simultaneously invoked by the database monitor.
RDM$BIND_OPTIMIZE_AIJ_RECLEN RDB_BIND_OPTIMIZE_AIJ_RECLEN	Allows you to tune the performance of the RMU Optimize command by specifying an average length for the AIJ record.
RDM$BIND_RCACHE_INSERT_ENABLED RDB_BIND_RCACHE_INSERT_ENABLED	Indicates whether rows can be inserted into the row cache.
RDM$BIND_RCACHE_RCRL_COUNT RDB_BIND_RCACHE_RCRL_COUNT	Indicates the number of reserved row cache slots.
RDM$BIND_RCS_BATCH_COUNT RDB_BIND_RCS_BATCH_COUNT	Defines the number of rows that the row cache server (RCS) sweeps in a single batch.
RDM$BIND_RCS_CHECKPOINT RDB_BIND_RCS_CHECKPOINT	Indicates whether the row cache server (RCS) performs a checkpoint.
RDM$BIND_RCS_CKPT_BUFFER_CNT RDB_BIND_RCS_CKPT_BUFFER_CNT	Indicates the number of buffers to be examined as a single batch by the row cache server (RCS) process during a checkpoint operation.
RDM$BIND_RCS_LOG_FILE RDB_BIND_RCS_LOG_FILE	Defines a file name for the row cache server (RCS) log file.
RDM$BIND_RCS_MAX_COLD RDB_BIND_RCS_MAX_COLD	Indicates the number of marked records above which the row cache server (RCS) sweep starts.
RDM$BIND_RCS_MIN_COLD RDB_BIND_RCS_MIN_COLD	Indicates the number of unmarked records below which the row cache server (RCS) sweep completes.
RDM$BIND_RCS_SWEEP_INTERVAL RDB_BIND_RCS_SWEEP_INTERVAL	Indicates the amount of time, in minutes, between the RCS sweeps.
RDM$BIND_READY_AREA_SERIALLY RDB_BIND_READY_AREA_SERIALLY	Causes Oracle Rdb to grant lock requests for logical and physical areas in the order that the lock requests were made, which can prevent lock starvation.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDM$BIND_RUJ_ALLOC_BLKCNT RDB_BIND_RUJ_ALLOC_BLKCNT	Allows you to override the default value of the .ruj file. The .ruj file is created with 127 blocks by default.
RDM$BIND_RUJ_EXTEND_BLKCNT RDB_BIND_RUJ_EXTEND_BLKCNT	Allows you to pre-extend the .ruj files for each process using a database. The block count value can be defined between 0 and 9999 (not inclusive) with a default of 100.
RDM$BIND_SNAP_QUIET_POINT RDB_BIND_SNAP_QUIET_POINT	Allows you to control whether snapshot transactions hold the quiet-lock and stall database or AIJ backups from operating.
RDM$BIND_STATS_AIJ_ARBS_PER_IO RDB_BIND_STATS_AIJ_ARBS_PER_IO	Allows you to override the default value of AIJ request blocks per AIJ I/O. The default is 2 blocks.
RDM$BIND_STATS_AIJ_BKGRD_ARB_RATIO RDB_BIND_STATS_AIJ_BKGRD_ARB_RATIO	Allows you to override the default value for the background AIJ request block threshold. The default value is 50.
RDM$BIND_STATS_AIJ_BLKS_PER_IO RDB_BIND_STATS_AIJ_BLKS_PER_IO	Allows you to override the default value of blocks per AIJ I/O. The default value is 2.
RDM$BIND_STATS_AIJ_SEC_TO_EXTEND RDB_BIND_STATS_AIJ_SEC_TO_EXTEND	Allows you to override the default value of seconds to AIJ extend. The default value is 60.
RDM$BIND_STATS_BTR_FETCH_DUP_RATIO RDB_BIND_STATS_BTR_FETCH_DUP_RATIO	Allows you to override the default value of the B-tree duplicate fetch threshold. The default threshold is 15.
RDM$BIND_STATS_BTR_LEF_FETCH_RATIO RDB_BIND_STATS_BTR_LEF_FETCH_RATIO	Allows you to override the default value of the B-tree leaf node fetch threshold. The default threshold is 25.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDM$BIND_STATS_DBR_RATIO RDB_BIND_STATS_DBR_RATIO	Allows you to override the default value of the DBR invocation threshold. The default threshold is 15.
RDM$BIND_STATS_ENABLED RDB_BIND_STATS_ENABLED	Allows you to enable or disable the writing of database statistics for a process. By default, writing is enabled.
RDM$BIND_STATS_FULL_BACKUP_INTRVL RDB_BIND_STATS_FULL_BACKUP_INTRVL	Allows you to override the full database backup threshold. The default threshold is 6.
RDM$BIND_STATS_GB_IO_SAVED_RATIO RDB_BIND_STATS_GB_IO_SAVED_RATIO	Allows you to override the GB IO-saved default threshold. The default threshold is 85.
RDM$BIND_STATS_GB_POOL_HIT_RATIO RDB_BIND_STATS_GB_POOL_HIT_RATIO	Allows you to override the GB pool hit default threshold. The default threshold is 85.
RDM$BIND_STATS_LB_PAGE_HIT_RATIO RDB_BIND_STATS_LB_PAGE_HIT_RATIO	Allows you to override the LB/AS page hit default threshold. The default is 75.
RDM$BIND_STATS_MAX_HASH_QUE_LEN RDB_BIND_STATS_MAX_HASH_QUE_LEN	Allows you to override the hash table queue length default threshold. The default threshold is 2 rows.
RDM$BIND_STATS_MAX_LOCK_STALL RDB_BIND_STATS_MAX_LOCK_STALL	Allows you to override the lock stall default threshold. The default threshold is 2 seconds.
RDM$BIND_STATS_MAX_TX_DURATION RDB_BIND_STATS_MAX_TX_DURATION	Allows you to override the transaction duration default threshold. The default value is 15.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDM$BIND_STATS_PAGES_CHECKED_RATIO RDB_BIND_STATS_PAGES_CHECKED_RATIO	Allows you to override the pages checked default threshold. The default threshold is 10 pages.
RDM$BIND_STATS_RECS_FETCHED_RATIO RDB_BIND_STATS_RECS_FETCHED_RATIO	Allows you to override the records fetched default threshold. The default threshold is 20 records.
RDM$BIND_STATS_RECS_STORED_RATIO RDB_BIND_STATS_RECS_STORED_RATIO	Allows you to override the records stored default threshold. The default threshold is 20 records.
RDM$BIND_STATS_RUJ_SYNC_IO_RATIO RDB_BIND_STATS_RUJ_SYNC_IO_RATIO	Allows you to override the synchronous RUJ I/O default threshold. The default threshold is 10.
RDM$BIND_STATS_VERB_SUCCESS_RATIO RDB_BIND_STATS_VERB_SUCCESS_RATIO	Allows you to override the verb success default threshold. The default threshold is 25.
RDM$BIND_SYSTEM_BUFFERS_ENABLED Not applicable on Digital UNIX or OpenVMS VAX	Indicates whether system space global sections are used.
RDM$BIND_TSN_INTERVAL RDB_BIND_TSN_INTERVAL	Indicates the number of transactions to be allocated as a single batch when the commit to journal optimization feature is enabled.
RDM$BIND_VM_SEGMENT RDB_BIND_VM_SEGMENT	Allocates the number of bytes necessary to avoid memory fragmentation.
RDM$BUGCHECK_DIR RDB_BUGCHECK_DIR	Allows you to redirect bugcheck files from the default directory to another location. This can be useful if the current default login directory does not have enough space for bugcheck dump files.
RDM$BUGCHECK_IGNORE_FLAGS RDB_BUGCHECK_IGNORE_FLAGS	Allows you to reduce the size of the bugcheck dump files.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDM$MAILBOX_CHANNEL Not applicable on Digital UNIX	Contains the node-specific address of the database monitor mailbox. This address is used by processes to communicate with the appropriate database monitor.
RDM$MONITOR RDB_MONITOR	Defines the device and directory where the Oracle Rdb monitor log file is to reside. The logical value should not include a file name specification. The directory location defined by the value of the logical name or configuration parameter is tested and used only by the RMONSTART.COM command file.
RDM$MON_USERNAME Not applicable on Digital UNIX	Designates the name of the user whose quotas the monitor process, upon startup, is to inherit. This logical name helps you avoid exceeded quota errors that can occur at startup time if global buffers are enabled.
RDMS$AUTO_READY RDB_AUTO_READY	Allows a process requesting a logical area lock in CR mode to obtain the lock in CU mode if the process already holds a carry-over lock in CU mode for the logical area.
RDMS$BIND_OUTLINE_FLAGS RDB_BIND_OUTLINE_FLAGS	Causes Oracle Rdb to ignore query outlines.
RDMS$BIND_OUTLINE_MODE RDB_BIND_OUTLINE_MODE	When multiple outlines exist for a query, this logical name is used to select the outline to use.
RDMS$BIND_PRESTART_TXN RDB_BIND_PRESTART_TXN	Allows you to establish the default setting for prestarted transactions outside of an application.
RDMS$BIND_QG_CPU_TIMEOUT RDB_BIND_QG_CPU_TIMEOUT	Restricts the amount of CPU time used to optimize a query for execution. If a user enters a query and the elapsed CPU time specified by this value is exceeded, the user receives an error message and the query is aborted.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDMS$BIND_QG_REC_LIMIT RDB_BIND_QG_REC_LIMIT	Establishes a process or system limit on the number of rows a query returns. If a user enters a query and the returned rows exceed the limit set by this value, the user receives an error message and the query is aborted.
RDMS$BIND_QG_TIMEOUT RDB_BIND_QG_TIMEOUT	Establishes a system limit on the amount of time the optimizer spends compiling a query. If a user enters a query and the elapsed time specified by this value is exceeded, the user receives an error message and the query is aborted.
RDMS$BIND_SEGMENTED_STRING_BUFFER RDB_BIND_SEGMENTED_STRING_BUFFER	Allows you to reduce the overhead of I/O operations when you manipulate segmented strings.
RDMS$BIND_SEGMENTED_STRING_COUNT RDB_BIND_SEGMENTED_STRING_COUNT	Specifies the allocation size, expressed as the number of entries, in the segmented string ID list; the ID list is used to materialize and manipulate segmented strings for a table row. Defining this logical name or configuration parameter can help to avoid a problem in which an import operation fails and the process loops.
RDMS$BIND_SEGMENTED_STRING_DBKEY_SCOPE RDB_BIND_SEGMENTED_STRING_DBKEY_SCOPE	Used to indicate whether the dbkey of a modified segmented string may be reused by the process.
RDMS$BIND_SORT_WORKFILES RDB_BIND_SORT_WORKFILES	Specifies how many work files SORT is to use if work files are required.
RDMS$BIND_VALIDATE_CHANGE_FIELD RDB_BIND_VALIDATE_CHANGE_FIELD	Causes the SQL ALTER DOMAIN statement to validate data records and convert them to the new metadata definitions.
RDMS$BIND_WORK_FILE RDB_BIND_WORK_FILE	Allows you to reduce the overhead of disk I/O for matching operations when used in conjunction with RDMS$BIND_WORK_VM or RDB_BIND_WORK_VM.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDMS$BIND_WORK_VM RDB_BIND_WORK_VM	Allows you to reduce the overhead of disk I/O for matching operations that use temporary tables by letting you specify the amount of virtual memory (VM) that will be allocated to your process for use in matching operations.
RDMS$DEBUG_FLAGS RDB_DEBUG_FLAGS	Allows you to examine database access strategies and the estimated cost of those strategies when your program is run.
RDMS$DEBUG_FLAGS_OUTPUT RDB_DEBUG_FLAGS_OUTPUT	Allows you to name an output file in which to collect the output from RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS when you run your program.
RDMS$DIAG_FLAGS RDB_DIAG_FLAGS	Can be used to provide assistance in locating erroneous queries.
RDMS$KEEP_PREP_FILES Not applicable on Digital UNIX	Causes the RDBPRE preprocessor to retain the intermediate macro (.mar) and language files. By default, these files are deleted when the preprocessor completes the processing of your source file.
RDMS$RUJ RDB_RUJ	Can be used to locate the .ruj file on a different disk and directory from the location of the .rdb file to reduce contention on the disk and directory used for the database.
RDMS$USE_OLD_CONCURRENCY RDB_USE_OLD_CONCURRENCY	Allows applications to use the isolation level behavior that was in effect for Oracle Rdb V4.1.
RDMS$USE_OLD_COST_MODEL RDB_USE_OLD_COST_MODEL	Causes the optimizer to not use workload or storage statistics.
RDMS$USE_OLD_COUNT_RELATION RDB_USE_OLD_COUNT_RELATION	Allows you to disable CREATE INDEX optimization for empty tables.
RDMS$USE_OLD_SEGMENTED_STRING RDB_USE_OLD_SEGMENTED_STRING	Retains the old format (chained) segmented strings as the default. Mixing of old and new format segmented strings is supported.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
RDMS$USE_OLD_UPDATE_RULES RDB_USE_OLD_UPDATE_RULES	Enforces old update rules for applications that use RDO. Oracle Rdb versions prior to Version 4.1 do not allow you to modify or erase records in a relation if the relation is joined with other relations. Use this logical name to preserve the RDO update behavior of Oracle Rdb versions prior to Version 4.1 until you can modify your applications.
RDMS$VALIDATE_ROUTINE RDB_VALIDATE_ROUTINE	Marks an invalid routine as valid. When a process assigns a value of 1 to RDMS$VALIDATE_ROUTINE or RDB_VALIDATE_ROUTINE, Oracle Rdb marks each invalid routine as valid when the process calls the procedure within a read/write transaction.
RDO$EDIT Not applicable on Digital UNIX	Indicates the system editor to be used to edit interactive RDO queries.
RDOINI Not applicable on Digital UNIX	Specifies the name of the file that contains the RDO initialization information. If the logical name exists, *and* the indicated file exists, RDO executes the commands in this file first, *before* displaying the RDO prompt and accepting input commands.
RMU$EDIT Not applicable on Digital UNIX	Indicates the system editor to be used to edit the Notepad in the Performance Monitor tools facility. The valid values are EDT, LSE, and TPU. The default editor is EDT.
SQL$DATABASE SQL_DATABASE	Specifies the database that SQL declares if you do not explicitly declare a database.
SQL$DISABLE_CONTEXT Not applicable on Digital UNIX	Disables the two-phase commit protocol. This is useful for turning off distributed transactions when you want to run batch-update transactions.
SQL$EDIT Not applicable on Digital UNIX	Indicates the system editor to be used to edit interactive SQL queries.

(continued on next page)

Table 2–6 (Cont.) Summary of Oracle Rdb Logical Names and Configuration Parameters

Logical Name Configuration Parameter	Function
SQLINI Not applicable on Digital UNIX	Specifies the name of the file that contains the SQL initialization information. If the logical name exists, *and* the indicated file exists, SQL executes the commands in this file first, *before* displaying the SQL prompt and accepting input commands.
SQL$KEEP_PREP_FILES SQL_KEEP_PREP_FILES	Causes the SQL precompiler or SQL module language compiler to retain the intermediate macro (.mar) and language files. By default, these files are deleted when the precompiler completes the processing of your source file.

2.3.1 Performance Monitor Defined Logicals Screen in the Character-Cell Interface

OpenVMS VAX OpenVMS Alpha

The Defined Logicals screen shows all the logical names currently accessible to the Performance Monitor, the name of the table in which they were defined, and the associated logical definition.

You access the Defined Logicals screen by selecting the Process Information option from the main menu, then selecting the Defined Logicals option from the submenu.

The displayed logical names are listed in suffix alphabetical order for quick review. That is, the logical names are sorted after removing the facility name (such as RDMS$, RDM$, SQL$). This method of sorting is very useful, because in most cases the facility name prefix is not known.

The Defined Logicals screen has two modes: brief or full. The brief display mode is the default. You select brief mode by typing B (displayed as Brief on the horizontal menu). When the Defined Logicals screen is in brief mode, only those logical names actually defined and accessible appear. The displayed table name is the actual table where the logical name resides, and the definition is the logical name's defined value. The screen is dynamically updated as new logical names are defined on the system. The following shows a Defined Logicals screen in brief mode:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 17-MAY-1996 14:46:12
Rate: 3.00 Seconds             Defined Logicals           Elapsed: 05:47:55.00
Page: 1 of 3         SQL_DISK1:[USER]MF_PERSONNEL.RDB;1            Mode: Online
--------------------------------------------------------------------------------
Logical.Name................ Table.Name......... Logical.Definition..........
SQL$DATABASE                 LNM$PROCESS_TABLE   SQL_PERSONNEL
RDM$MAILBOX_CHANNEL          LNM$SYSTEM_TABLE    MBA64:
RDM$MONITOR                  LNM$SYSTEM_TABLE    SYS$SYSROOT:[SYSEXE]









--------------------------------------------------------------------------------
Exit Full Help Menu >next_page <prev_page Set_rate Write !
```

You select full mode by typing F (displayed as Full on the horizontal menu). When the Defined Logicals screen is in full mode, all known logical names are displayed, whether the logical name is defined or not. If the displayed logical name is defined, the table name and logical value are displayed as if the display were in brief mode. If the logical name is not defined, the table name contains the name of the table where the logical name should reside; no definition is displayed. The full mode is useful for checking the spelling of logical names and ensuring the logical name is defined in the proper table. The following shows a Defined Logicals screen in full mode:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 17-MAY-1996 14:48:03
Rate: 3.00 Seconds            Defined Logicals           Elapsed: 05:50:17.67
Page: 1 of 5         SQL_DISK1:[USER]MF_PERSONNEL.RDB;1            Mode: Online
--------------------------------------------------------------------------------
Logical.Name................ Table.Name......... Logical.Definition..........
RDMS$AUTO_READY              LNM$FILE_DEV
RDM$BIND_ABS_QUIET_POINT     LNM$FILE_DEV
RDM$BIND_ABW_ENABLED         LNM$FILE_DEV
RDM$BIND_AIJ_IO_MAX          LNM$FILE_DEV
RDM$BIND_AIJ_STALL           LNM$FILE_DEV
RDM$BIND_APF_DEPTH           LNM$FILE_DEV
RDM$BIND_APF_ENABLED         LNM$FILE_DEV
RDM$BIND_BATCH_MAX           LNM$FILE_DEV
RDM$BIND_BUFFERS             LNM$FILE_DEV
RDM$BIND_CKPT_TRANS_INTERVAL  LNM$FILE_DEV
RDM$BIND_CLEAN_BUF_CNT       LNM$FILE_DEV
RDM$BIND_COMMIT_STALL        LNM$FILE_DEV
RDM$BIND_DAPF_DEPTH_BUF_CNT   LNM$FILE_DEV
RDM$BIND_DAPF_ENABLED        LNM$FILE_DEV
RDM$BIND_DAPF_START_BUF_CNT   LNM$FILE_DEV
RDMS$BIND_EXEC_STACK_SIZE    LNM$FILE_DEV
RDMS$BIND_EXT_FILE_NAME_ONLY  LNM$FILE_DEV
--------------------------------------------------------------------------------
Brief Exit Help Menu >next_page <prev_page Set_rate Write !
```

The Defined Logicals screen shows the list of all logical names currently accessible to the Performance Monitor; the screen is unable to show logical names defined in another process' tables.

The Defined Logicals screen shows the values of logical names; it does not allow you to modify their definitions.

The output from the Defined Logicals screen is not written to the output file; therefore, the output cannot be replayed using an input file.

Due to screen width limitations, only the first 28 characters of the logical name are displayed.

If you are using a multiversion product, the logical names displayed do not contain the numerical suffix. For example, if the multiversion variant of Oracle Rdb Version 6.1 is used, the 61 suffix does not appear for any of the logical names in the Defined Logicals screen. ♦

2.4 Oracle Trace for OpenVMS

OpenVMS VAX OpenVMS Alpha

Oracle Trace is a product that collects, reports, and displays event-based data gathered from Oracle Rdb. If Oracle Trace is installed on your system, you can use it to collect detailed information about Oracle Rdb and applications that have been instrumented for Oracle Trace.

Oracle Rdb has been instrumented to log Oracle Trace data. **Instrumenting** is the process of adding Oracle Trace system service routine calls to application code. These Oracle Trace system service routines are placed so that they collect data for Oracle Rdb events as they execute.

Section 2.4.1 explains how Oracle Rdb is instrumented for Oracle Trace. Section 2.4.2 provides an overview of how to create a selection, how to schedule a collection, how to generate a report, and how to display event data interactively. This Oracle Trace information should be used with the Oracle Trace documentation.

2.4.1 Oracle Rdb Instrumentation

Oracle Trace collects data for two kinds of events: point events such as REQUEST_BLR events, and duration events such as TRANSACTION events.

Specific kinds of information (referred to by Oracle Trace as *items*) are associated with each event. Oracle Trace can potentially collect many kinds of items for each event.

Table 2–7 describes the Oracle Rdb events instrumented for Oracle Trace.

Table 2–7 Oracle Rdb Events

Event	Description	Type of Event
DATABASE	Records each attach to a database.	Point
REQUEST_ACTUAL	Records the execution of a single instance of a user's request.	Duration
REQUEST_BLR	Records the binary language representation (BLR) for each request each time the request is executed.	Point
TRANSACTION	Records the start and end of a transaction in the database.	Duration

Each item described in this chapter is from either the set of standard Oracle Trace resource utilization items or from the set of items specific for Oracle Rdb. Table 2–8 lists the standard resource utilization items that Oracle Trace collects by default. The items listed in Table 2–8 are also referred to as the RESOURCE_ITEMS group. To help you collect only the information you are interested in, Oracle Trace organizes data items into groups, such as the following:

- RESOURCE_ITEMS

  See Table 2–8 for more information on the RESOURCE_ITEMS group.

- AREA_ITEMS

  See Table 2–10 for more information on the AREA_ITEMS group.
- DATABASE_ITEMS

  See Table 2–11 for more information on the DATABASE_ITEMS group.
- RDB_CROSS_FAC

  See Table 2–12 for more information on the RDB_CROSS_FAC group.

Table 2–8 Resource Utilization Items

Item	Description	Data Type	Usage
BIO	Number of buffered I/O operations	Longword	Counter
CPU	Total amount of CPU time	Longword	Counter
CURRENT_PRIO	Current priority of the process	Word	Level
DIO	Number of direct I/O operations	Longword	Counter
PAGEFAULT_IO	Number of hard page faults (that is, page faults to or from the disk)	Longword	Counter
PAGEFAULTS	Total number of hard and soft page faults	Longword	Counter
VIRTUAL_SIZE	Number of virtual pages currently mapped for the process	Longword	Level
WS_GLOBAL	Number of pages in the working set that are globally shared among processes on the system	Longword	Level
WS_PRIVATE	Number of pages in the working set that are private to the process	Longword	Level
WS_SIZE	Current working set size of the process	Longword	Level

Table 2–9 describes the items that are specific to Oracle Rdb.

Table 2–9 Oracle Rdb Data Items

Item	Description	Data Type	Usage
AIJ_WRITES	Number of write-QIOs issued to the database after-image journal (.aij) file. If after-image journaling is not enabled for the database, this statistic will be zero. This operation writes after-image records to the .aij file to facilitate roll-forward recovery (RMU Recover).	Longword	Counter
BLR	Binary language representation of the query being executed.	ASCII	Text Nonprintable
BUFFER_READS	Number of logical page requests for data (or snapshot) pages. These requests are calls to the PIO subsystem of Oracle Rdb. The page requests do not necessarily result in a physical read or write, because the requested page is often still cached in the database page buffer pool.	Longword	Counter
CLIENT_PC	Value of the program counter (PC) of the application program when a call was made to Oracle Trace.	Longword	Level
COMP_STATUS	OpenVMS completion status.	Longword	Level
CROSS_FAC_2	The server index value, which uniquely describes which ACMS procedure server generated the transaction.	Longword	Index
CROSS_FAC_3	Value that uniquely describes the form or report that generated the transaction. Used by any 4GL that does not call the internal Oracle Rdb interface.	Longword	Index
CROSS_FAC_7	The procedure index value, which uniquely describes which ACMS procedure generated the transaction.	Longword	Index
CROSS_FAC_14	Can be used by any application to uniquely describe its event that generated the transaction.	Longword	Index
DB_NAME	Name of the database.	ASCII	Text

(continued on next page)

Table 2–9 (Cont.) Oracle Rdb Data Items

Item	Description	Data Type	Usage
DBS_READS	Number of read-QIOs issued to the database storage area and snapshot files. This operation reads database pages from the database.	Longword	Counter
DBS_WRITES	Number of write-QIOs issued to the database storage area and snapshot files. This operation writes modified database pages back to the database.	Longword	Counter
D_FETCH_RET	The number of synchronous data page requests where read privileges are requested for the data page.	Longword	Counter
D_FETCH_UPD	The number of synchronous data page requests where update as well as read privileges are requested for the data page.	Longword	Counter
D_LB_ALLOK	The number of times the requested data page was found in the user's local buffer pool and the user already had the needed locks on the page.	Longword	Counter
D_LB_GBNEEDLOCK	The number of times the requested data page was found in the user's allocate set and the user held sufficient locks to satisfy the request. But because of global buffers, additional locking was needed to verify the version was correct. The version was correct, so the extra locking overhead was solely because global buffers were used.	Longword	Counter
D_LB_NEEDLOCK	The number of times the requested data page was found in the user's local buffer pool, but additional locking was required to lock the page in the needed mode.	Longword	Counter

(continued on next page)

Table 2–9 (Cont.) Oracle Rdb Data Items

Item	Description	Data Type	Usage
D_LB_OLDVER	The number of times the requested data page was found in the user's local buffer pool, but it was an obsolete version of the page (because the page was changed by another user since it was read into the user's local buffer), which required the page to be read again from disk.	Longword	Counter
D_GB_NEEDLOCK	The number of times the correct version of the requested data page was found in the user's allocate set but additional locking was required to lock the page in the needed mode.	Longword	Counter
D_GB_OLDVER	The number of times the requested data page was found in the user's allocate set, but it was an obsolete version of the page (that is, the page has been changed by another user since it was read into the requestor's allocate set), which required the page to be read again from disk.	Longword	Counter
D_NOTFOUND_IO	The number of times the requested data page was not found in the buffer pool and had to be read in from disk.	Longword	Counter
D_NOTFOUND_SYN	The number of times the requested data page was not found in the buffer pool but could be synthesized into the pool without being read from disk.	Longword	Counter
FREE_VM_BYTES	Number of virtual memory bytes returned by Oracle Rdb to the process-local virtual memory pool. The difference between total GET_VM_BYTES and FREE_VM_BYTES indicates the amount of virtual memory still in use.	Longword	Counter
GET_VM_BYTES	Number of virtual memory bytes requested by Oracle Rdb from the process-local virtual memory pool. The difference between total GET_VM_BYTES and FREE_VM_BYTES indicates the amount of virtual memory still in use.	Longword	Counter

(continued on next page)

Table 2–9 (Cont.) Oracle Rdb Data Items

Item	Description	Data Type	Usage
GLOBAL_TID	Transaction identifier used in distributed transactions.	ASCII	Private Nonprintable
IMAGE_NAME	Name of the node and image making dispatch calls.	ASCII	Text
IO_EXT_BLKCNT	Number of storage area block extends.	Longword	Counter
IO_EXTEND_CNT	Number of storage area extends.	Longword	Counter
IO_EXTEND_STALL	Number of storage area extend stalls.	Longword	Counter
IO_READ_BLKCNT	Number of storage area block reads.	Longword	Counter
IO_READ_CNT	Number of storage area reads.	Longword	Counter
IO_READ_STALL	Number of storage area read stalls.	Longword	Counter
IO_WRITE_BLKCNT	Number of storage area block writes.	Longword	Counter
IO_WRITE_CNT	Number of storage area writes.	Longword	Counter
IO_WRITE_STALL	Number of storage area write stalls.	Longword	Counter
LOCK_MODE	OpenVMS lock mode.	Longword	Level
LOCK_RELS	Number of $DEQ lock requests to release an existing lock. These requests always succeed. The number of outstanding locks can be determined by subtracting REQ_NOT_QUEUED, REQ_DEADLOCKS, and LOCK_RELS from LOCK_REQS.	Longword	Counter
LOCK_REQS	Number of $ENQ lock requests for new locks. A request is included in this count regardless of whether or not it succeeds.	Longword	Counter
LOCK_STALL_TIME	Total time (in hundredths of a second) spent by all users waiting for locks within the scope of a transaction. This statistic gives a relative measure of work lost due to lock conflicts.	Longword	Counter

(continued on next page)

Table 2–9 (Cont.) Oracle Rdb Data Items

Item	Description	Data Type	Usage
PROM_DEADLOCKS	Number of stalled $ENQ lock requests to promote an existing lock to a higher lock mode that ultimately resulted in a deadlock. Most deadlocks are resolved and retried by Oracle Rdb transparently to the application program. Therefore, this number does not necessarily reflect the number of deadlocks reported to the application program.	Longword	Counter
REQ_DEADLOCKS	Number of stalled $ENQ lock requests that ultimately resulted in a deadlock. Most deadlocks are resolved and retried by Oracle Rdb transparently to the application program. Therefore, this number does not necessarily reflect the number of deadlocks reported to the application program.	Longword	Counter
REQ_ID	Unique identification of a request to a database.	Longword	Level
REQ_NOT_ QUEUED	Number of $ENQ lock requests for new locks that were rejected immediately because of a lock conflict. Oracle Rdb often requests a lock without waiting, and when a conflict is detected, resorts to a secondary locking protocol to avoid unnecessary deadlocks. This number is one measure of lock contention.	Longword	Counter
REQ_STALLS	Number of $ENQ lock requests for new locks that were stalled because of a lock conflict. A request is included in this count regardless of whether or not it succeeds. This number is one measure of lock contention.	Longword	Counter
REQUEST_COUNT	Number of times a request executes within a transaction.	Longword	Level

(continued on next page)

Table 2–9 (Cont.) Oracle Rdb Data Items

Item	Description	Data Type	Usage
ROOT_READS	Number of read-QIOs issued to the database root (.rdb) file. Oracle Rdb reads the root file when a new user attaches to the database and when a root file control block needs to be refreshed because of database activity on another VMScluster node.	Longword	Counter
ROOT_WRITES	Number of write-QIOs issued to the database root (.rdb) file. Oracle Rdb writes the root file when a user issues a COMMIT or ROLLBACK statement. Other events may also cause the root file to be updated.	Longword	Counter
RUJ_READS	Number of read-QIOs issued to the database recovery unit journal (.ruj) files. This operation reads before-image records from the .ruj files to roll back a verb or a transaction.	Longword	Counter
RUJ_WRITES	Number of write-QIOs issued to the database recovery unit journal (.ruj) files. This operation writes before-image records to the .ruj files in case a verb or transaction must be rolled back. Before-images must be written to the .ruj files before the corresponding database page can be written back to the database.	Longword	Counter
STREAM_ID	Identification of a unique attach to a database. It identifies a database user from the database's point of view.	Longword	Level
S_FETCH_RET	The number of synchronous SPAM page requests where read privileges are requested for the SPAM page.	Longword	Counter
S_FETCH_UPD	The number of synchronous SPAM page requests where update as well as read privileges are requested for the SPAM page.	Longword	Counter

(continued on next page)

Table 2–9 (Cont.) Oracle Rdb Data Items

Item	Description	Data Type	Usage
S_LB_ALLOK	The number of times the requested SPAM page was found in the user's local buffer pool and the user already had the needed locks on the page.	Longword	Counter
S_LB_GBNEEDLOCK	The number of times the requested SPAM page was found in the user's allocate set and the user held sufficient locks to satisfy the request. But because of global buffers, additional locking was needed to verify the version was correct. The version was correct, so the extra locking overhead was solely because global buffers were used.	Longword	Counter
S_LB_NEEDLOCK	The number of times the requested SPAM page was found in the user's local buffer pool, but additional locking was required to lock the page in the needed mode.	Longword	Counter
S_LB_OLDVER	The number of times the requested SPAM page was found in the user's local buffer pool, but it was an obsolete version of the page (because the page was changed by another user since it was read into the user's local buffer), which required the page to be read again from disk.	Longword	Counter
S_GB_NEEDLOCK	The number of times the correct version of the requested SPAM page was found in the user's allocate set but additional locking was required to lock the page in the needed mode.	Longword	Counter
S_GB_OLDVER	The number of times the requested SPAM page was found in the user's allocate set, but it was an obsolete version of the page (that is, the page had been changed by another user since it was read into the requestor's allocate set), which required the page to be read again from disk.	Longword	Counter

(continued on next page)

Table 2–9 (Cont.) Oracle Rdb Data Items

Item	Description	Data Type	Usage
S_NOTFOUND_IO	The number of times the requested SPAM page was not found in the buffer pool and had to be read in from disk.	Longword	Counter
S_NOTFOUND_SYN	The number of times the requested page was not found in the buffer pool but could be synthesized into the pool without being read from disk.	Longword	Counter
TRANS_ID	Unique identification number associated with a transaction.	ASCII	Private Nonprintable

The AREA_ITEMS group is specific to Oracle Rdb. Table 2–10 shows the items associated with the AREA_ITEMS group. There is one set of nine area items for each storage area in a database. Thus, if your database has 10 storage areas, it will have 90 area items.

Table 2–10 Items Associated with the AREA_ITEMS Group

IO_EXT_BLKCNT	IO_EXTEND_CNT	IO_EXTEND_STALL
IO_READ_BLKCNT	IO_READ_CNT	IO_READ_STALL
IO_WRITE_BLKCNT	IO_WRITE_CNT	IO_WRITE_STALL

The DATABASE_ITEMS group is specific to Oracle Rdb. Table 2–11 shows the items associated with the DATABASE_ITEMS group.

Table 2–11 Items Associated with the DATABASE_ITEMS Group

AIJ_WRITES	BUFFER_READS	DBS_READS
DBS_WRITES	D_FETCH_RET	D_FETCH_UPD
D_LB_ALLOK	D_LB_GBNEEDLOCK	D_LB_NEEDLOCK
D_LB_OLDVER	D_GB_NEEDLOCK	D_GB_OLDVER
D_NOTFOUND_IO	D_NOTFOUND_SYN	S_FETCH_RET
S_FETCH_UPD	S_LB_ALLOK	S_LB_GBNEEDLOCK
S_LB_NEEDLOCK	S_LB_OLDVER	S_GB_NEEDLOCK
S_GB_OLDVER	S_NOTFOUND_IO	S_NOTFOUND_SYN
FREE_VM_BYTES	GET_VM_BYTES	LOCK_RELS
LOCK_REQS	LOCK_STALL_TIME	PROM_DEADLOCKS
REQ_DEADLOCKS	REQ_NOT_QUEUED	REQ_STALLS
ROOT_READS	ROOT_WRITES	RUJ_READS
RUJ_WRITES		

Oracle Trace lets you relate events among facilities. Programmers often relate events among two or more facilities to gain an understanding of the full context in which the events occurred. Programmers relate events among applications in two ways. The most common way is to explicitly pass data items between applications through an application programming interface (API). However, sometimes facilities cannot change the existing API, or cannot pass data items through it, as with SQL. To accommodate these applications, Oracle Trace provides the cross-facility feature. Programmers can relate events among facilities using the cross-facility feature and the cross-facility items specific to Oracle Rdb that appear in Table 2–12.

Table 2–12 Items Associated with the RDB_CROSS_FAC Group

CROSS_FAC_2	CROSS_FAC_3	CROSS_FAC_7
CROSS_FAC_14		

Table 2–13 through Table 2–17 list the item for each event in the collection classes. Table 2–18 through Table 2–21 show the available events for each collection class.

Table 2–13 lists the items associated with the DATABASE event.

Table 2–13 Items Associated with the DATABASE Event

CLIENT_PC	DB_NAME	IMAGE_FILE_NAME
STREAM_ID		

Table 2–14 lists the items associated with the REQUEST_ACTUAL event.

Table 2–14 Items Associated with the REQUEST_ACTUAL Event

CLIENT_PC	COMP_STATUS	DATABASE_ITEMS[1]
REQUEST_OPER	REQ_ID	RESOURCE_ITEMS[2]
STREAM_ID	TRANS_ID	

[1] See Table 2–11 for a list of items in the DATABASE_ITEMS group.

[2] See Table 2–8 for a list of items in the RESOURCE_ITEMS group.

Table 2–15 lists the items associated with the REQUEST_BLR event.

Table 2–15 Items Associated with the REQUEST_BLR Event

BLR	CLIENT_PC	REQ_ID
STREAM_ID	TRANS_ID	

Oracle Trace groups items and events into classes so that you can limit your collections to the information that you need. This chapter describes the following classes:

- PERFORMANCE

  Table 2–18 provides more information on the PERFORMANCE class.

- PERFORMANCE_NO_CF

  Table 2–19 provides more information on the PERFORMANCE_NO_CF class.

- RDBEXPERT

  Table 2–20 provides more information on the RDBEXPERT class.

- RDBEXPERT_NO_CF

  Table 2–21 provides more information on the RDBEXPERT_NO_CF class.

Table 2–16 lists the items associated with the TRANSACTION event in the PERFORMANCE and RDBEXPERT classes.

Table 2–16 Items Associated with the TRANSACTION Event in the PERFORMANCE and RDBEXPERT Classes

AREA_ITEMS[1]	CLIENT_PC	DATABASE_ITEMS[2]
GLOBAL_TID	LOCK_MODE	RDB_CROSS_FAC[3]
RESOURCE_ITEMS[4]	STREAM_ID	TRANS_ID

[1] See Table 2–10 for a list of items in the AREA_ITEMS group.

[2] See Table 2–11 for a list of items in the DATABASE_ITEMS group.

[3] See Table 2–12 for a list of items in the RDB_CROSS_FAC group.

[4] See Table 2–8 for a list of items in the RESOURCE_ITEMS group.

Table 2–17 lists the items associated with the TRANSACTION event in the PERFORMANCE_NO_CF and RDBEXPERT_NO_CF classes.

Table 2–17 Items Associated with the TRANSACTION Event in the PERFORMANCE_NO_CF and RDBEXPERT_NO_CF Classes

CLIENT_PC	DATABASE_ITEMS[1]	GLOBAL_TID
LOCK_MODE	RESOURCE_ITEMS[2]	STREAM_ID
TRANS_ID		

[1] See Table 2–11 for a list of items in the DATABASE_ITEMS group.

[2] See Table 2–8 for a list of items in the RESOURCE_ITEMS group.

Table 2–18 lists the events and items that make up the PERFORMANCE collection class. These events and items have been selected for their importance in understanding the performance characteristics of Oracle Rdb applications. The PERFORMANCE class is the default collection class for Oracle Rdb.

Table 2–18 Events and Items Available in the PERFORMANCE Class for Oracle Rdb

Event	Event Type	Items
DATABASE	Point	See Table 2–13.
REQUEST_ACTUAL	Duration	See Table 2–14.
TRANSACTION	Duration	See Table 2–16.

Table 2–19 lists the events and items that make up the PERFORMANCE_NO_CF collection class. The PERFORMANCE_NO_CF collection class is used when you are not interested in cross-referencing client data and Oracle Rdb data.

Table 2–19 Events and Items Available in the PERFORMANCE_NO_CF Class for Oracle Rdb

Event	Event Type	Items
DATABASE	Point	See Table 2–13.
REQUEST_ACTUAL	Duration	See Table 2–14.
TRANSACTION	Duration	See Table 2–17.

Table 2–20 lists the events and items that make up the RDBEXPERT collection class. These events and items have been selected for their importance in understanding the workload characteristics of Oracle Rdb applications.

Table 2–20 Events and Items Available in the RDBEXPERT Class for Oracle Rdb

Event	Event Type	Items
DATABASE	Point	See Table 2–13.
REQUEST_ACTUAL	Duration	See Table 2–14.
TRANSACTION	Duration	See Table 2–16.
REQUEST_BLR	Point	See Table 2–15.

Table 2–21 lists the events and items that make up the RDBEXPERT_NO_CF collection class. The RDBEXPERT_NO_CF collection class is used when you are not interested in cross-referencing client data and Oracle Rdb data.

Table 2–21 Events and Items Available in the RDBEXPERT_NO_CF Class for Oracle Rdb

Event	Event Type	Items
DATABASE	Point	See Table 2–13.
REQUEST_ACTUAL	Duration	See Table 2–14.
TRANSACTION	Duration	See Table 2–17.
REQUEST_BLR	Point	See Table 2–15.

By default, Oracle Trace can collect data from the full set of events and items defined for a facility. Another collection class available for Oracle Rdb is the ALL collection class. For the current release of Oracle Rdb, the ALL class contains the same events and items as the RDBEXPERT class. However, while events or items added in future releases will, by definition, be added to the ALL class, they may not be included in the RDBEXPERT class.

2.4.2 Overview of Using Oracle Trace

You can use Oracle Trace to do the following:

- Create a selection
- Start a collection
- Stop a collection
- Display event data
- Format and obtain reports on event data

Section 2.4.2.1 through Section 2.5.3 of this manual and the Oracle Trace documentation provide more information on using Oracle Trace for these tasks.

2.4.2.1 Creating a Selection

This section provides an overview of creating a selection. A complete description of creating a selection can be found in the Oracle Trace documentation.

The Oracle Rdb instrumentation is very general and can provide data for a variety of uses, such as performance tuning, input to Oracle Expert for Rdb, and resource usage evaluation. You can tailor Oracle Trace collections for your specific needs by using a facility selection to limit your collection to Oracle Rdb and to the specific Oracle Rdb events and items that interest you. If you do not use a facility selection, Oracle Trace will collect data for all the applications on your system that are instrumented for Oracle Trace and all their associated data items.

Use the CREATE SELECTION command to create a facility selection that consists of:

- Selection name
- List of facilities for which to collect data
- Class of data to collect for each facility

The following example defines the facility selection MY_SELECTION to collect the default data for Oracle Rdb:

```
$ COLLECT CREATE SELECTION MY_SELECTION /FACILITY=RDBVMS
```

If more than one version of Oracle Rdb is available, by default Oracle Trace uses the version of Oracle Rdb with the latest creation date.

2.4.2.2 Scheduling Data Collection

You must schedule data collection on your system by using the schedule collection command before Oracle Trace can begin to collect information about Oracle Rdb. The scheduling collection specifications include:

- Output file or files for the collected data
- Start and end times (or alternately, the duration)
- Facility selection to use
- Scope of the collection (the entire cluster or just the local node)

The following example schedules the collection MY_COLLECTION to begin at 11:00 A.M. and end at noon on the current day. The collection uses the facility selection SELECT_ALL and runs on the local node. Oracle Trace stores the collected data in the file MY_DATA.DAT in your default device and directory.

```
$ COLLECT SCHEDULE COLLECTION MY_COLLECTION MY_DATA.DAT -
_$ /SELECTION=SELECT_ALL -
_$ /BEGINNING=11:00 /ENDING=12:00 -
_$ /NOCLUSTER /COLLECTION_FILES=(PROTECTION=(W:RW))
```

```
%EPC-S-SCHED, Data collection MY_COLLECTION is scheduled
```

Alternately, you can use the Duration qualifier in place of the Ending qualifier. You must specify the duration as a relative OpenVMS time. For example:

```
$ COLLECT SCHEDULE COLLECTION MY_COLLECTION MY_DATA.DAT -
_$ /SELECTION=MY_SELECTION -
_$ /BEGINNING=11:00 /DURATION="1:" -
_$ /NOCLUSTER /COLLECTION_FILES=(PROTECTION=(W:W))
```

```
%EPC-S-SCHED, Data collection MY_COLLECTION is scheduled
```

You can schedule either local or clusterwide data collection if you use the [NO]CLUSTER qualifier. By default, the SCHEDULE COLLECTION command schedules data collection to occur on every node in the cluster. To schedule data collection on a subset of the cluster, log in to each node that you want data collection to occur on and schedule local data collection on that node by specifying /NOCLUSTER. Note that on a standalone system the CLUSTER qualifier is ignored.

You can further limit the collection of data by using the REGISTRATION_ID qualifier of the Oracle Trace SCHEDULE COLLECTION command. See the Oracle Trace documentation for more information.

2.4.2.3 Stopping a Collection

If you do not specify a beginning and ending time, you can stop a collection with a command of the following format:

```
$ COLLECT CANCEL COLLECTION MY_COLLECTION /NOCONFIRM
```

You can also use this command to cancel a pending collection. ♦

2.5 Collecting Workload Information for Oracle Expert for Rdb

OpenVMS VAX ═══ OpenVMS Alpha ═══

You can use Oracle Trace to gather workload data from Oracle Rdb database applications. This data can be imported into Oracle Expert for Rdb for use in generating an optimized physical design for your database.

To collect the workload information, specify the RDBEXPERT or ALL classes in your facility selection. It makes no difference in the Oracle Expert for Rdb design process which collection class you use. The following example creates a facility selection named RDBVMS_WORK to collect workload data from Oracle Rdb:

```
$ COLLECT CREATE SELECTION RDBVMS_WORK/OPTIONS -
Option> FACILITY RDBVMS /CLASS=RDBEXPERT /VERSION="V7.0-0"
Option> Ctrl/z
```

Schedule a collection using the RDBVMS_WORK selection. You probably do not want to gather data from every Oracle Rdb application on the system, so use the REGISTRATION_ID qualifier to specify the image or images from which you want to collect information. The following example schedules a collection to gather data from the personnel application:

```
$ COLLECT SCHEDULE COLLECTION GET_WORKLOAD WORK.DAT -
_$ /SELECTION=RDBVMS_WORK -
_$ /BEGIN=09:00 /END=12:00 -
_$ /REGISTRATION_ID=( $100$DUA1:[TOOLS]PERSONNEL.EXE )
```

After data collection has ended, format the data collection file or files into an Oracle Rdb database. Oracle Expert for Rdb imports this database to use as a workload design element. Note that Oracle Expert for Rdb cannot import the workload information if you format your data collection files into an RMS file. The following example formats the data in WORK.DAT into an Oracle Rdb database named WORKLOAD_DATA.RDB:

```
$ COLLECT FORMAT WORK.DAT WORKLOAD_DATA
```

Use Oracle Expert for Rdb to import the workload data from the formatted database. Specify the name of the database file created by the Oracle Trace FORMAT command as the source of the workload to be imported.

See the Oracle Expert for Rdb documentation for more information.

2.5.1 Displaying Event-Data Interactively

You can use the Oracle Trace Monitor to display event-based data interactively from an open or previously closed data collection file. The Monitor has a Motif-based window interface that you can use to move from image-level to event-level information.

Use a command of the following format to start the Monitor:

```
$ COLLECT MONITOR SAMPLE_DATA.DAT
```

If you want to display information from a previously closed data collection file, you must use the REPLAY qualifier of the MONITOR command. Start the Monitor and display data from a previously collected data collection file by using a command similar to the following:

```
$ COLLECT MONITOR/REPLAY SAMPLE_DATA.DAT
```

You can stop the Monitor by selecting the EXIT option of the FILE menu at the top of the Process window.

Note

Oracle Corporation recommends that you use the FLUSH_INTERVAL qualifier on the SCHEDULE COLLECTION command when you plan to use the Monitor to view event-based data. The FLUSH_INTERVAL qualifier specifies in seconds how often Oracle Trace writes all process buffers to the data collection file. The FLUSH_INTERVAL qualifier ensures the accuracy of Monitor time and data displays. Oracle Corporation recommends a flush interval of 1 or 2 seconds, as shown in the following example:

```
$ COLLECT SCHEDULE COLLECTION SAMPLE_COLLECTION SAMPLE_DATA.DAT-
_$ /SELECTION=SAMPLE_SELECTION -
_$ /DURATION=:30 -
_$ /FLUSH_INTERVAL=(00:00:02) -
_$ /NOCLUSTER /COLLECTION_FILES=(PROTECTION=(W:RW))
```

2.5.2 Creating a Report Based on Collected Data

You can use Oracle Trace to produce reports based on the data collected from one or more Oracle Trace collections. To produce Oracle Trace reports:

1. Format and merge the data files.
2. Generate the report.

2.5.2.1 Formatting and Merging Data Files

Before Oracle Trace can generate a report on the collected data, you must use the FORMAT command to format the data in the data files into an Oracle Rdb database.[1] To format the data in the file MY_DATA.DAT and store the formatted data in an Oracle Rdb database named FORMATTED_DATA.RDB, enter:

```
$ COLLECT FORMAT MY_DATA.DAT FORMATTED_DATA
```

You can also use the FORMAT command to combine the data files from two or more collections into one formatted database. However, these collections must have been scheduled using the same facility selection.

You can combine data files in two ways: format several data files at once into the same Oracle Rdb database, or add a data file to an existing formatted database. To combine the data collected from several collections into a new formatted database named WEEK.RDB, enter:

```
$ COLLECT FORMAT MONDAY.DAT,TUESDAY.DAT,WEDNESDAY.DAT WEEK
```

To add the contents of the data file THURSDAY.DAT to the existing formatted database WEEK.RDB, enter:

```
$ COLLECT FORMAT /MERGE THURSDAY.DAT WEEK
```

See the description of the FORMAT command in the Oracle Trace documentation for a list of optimization parameters that improve the performance of formatting operations.

[1] An OpenVMS RMS formatted file is also available for users who plan to create their own reports based on the data. See the Oracle Trace documentation for information.

2.5.2.2 Generating a Report

Oracle Trace can generate tabular reports based on the data in an Oracle Rdb formatted database. Table 2–22 lists the three different types of reports that you can produce.

Table 2–22 Oracle Trace Reports

Type	Description
Summary report	Provides statistical evaluations of resource consumption by event. This information is useful for performance evaluation and capacity planning.
Detail report	Provides values of items collected for each event. This report is useful for debugging an application that you are instrumenting.
Frequency report	Provides a count of event occurrences for each interval within a collection. This report is useful for tracing the execution of an instrumented application.

In the simplest case, the REPORT command creates a Summary report on all of the data in the formatted database and displays the report on your current SYS$OUTPUT device (usually, your terminal). For example:

```
$ COLLECT REPORT FORMATTED_DATA.RDB
```

Note

Oracle Trace collects all occurrences of each event in your chosen collection class. You cannot limit a collection to individual events. You can, however, limit a report to specific events and items using the EVENTS and ITEMS qualifiers. Oracle Corporation recommends that you use these qualifiers when you generate Detail reports, which can be very large unless you specify the events and items that interest you.

Use the STATISTICS qualifier to specify the statistics Oracle Trace uses in the Summary report. This feature allows you to create customized reports. Valid statistics are:

- ALL
- COUNT (default)
- MAXIMUM

- MEAN
- MINIMUM
- STANDARD_DEVIATION
- TOTAL
- 95_PERCENTILE

The following example generates a Summary report based on the data collected for Oracle Rdb using the MAXIMUM, MINIMUM, and MEAN options of the STATISTICS qualifier:

```
$ COLLECT REPORT FORMATTED_DATA.RDB /FACILITY=RDBVMS -
_$ /TYPE=SUMMARY /STATISTICS=(MAXIMUM,MINIMUM,MEAN) -
_$ /OUTPUT=MY_SUMMARY.TXT
```

By default, Oracle Trace reports on the data collected for all the events contained in the formatted database. You can use the EVENTS qualifier to restrict the report to specific events. The following example generates a report based on the data collected for a subset of the events in Oracle Rdb:

```
$ COLLECT REPORT FORMATTED_DATA.RDB /OUTPUT=MY_REPORT.TXT -
_$ /FACILITY=RDBVMS -
_$ /EVENTS=(DATABASE,TRANSACTION)
```

2.5.3 Creating a Customized Report

You can create customized reports that provide information on specific events and items. The OPTIONS qualifier to the REPORT command allows you to specify characteristics for individual events and items. Example 2–7 shows the command to generate a report that uses a different report format for each event.

Example 2–7 Using Reporting Options to Generate a Customized Report

```
COLLECT REPORT MY_DATABASE -
        /SINCE = 1-JAN-1989 -
        /WIDTH = 80 -
        /TYPE = SUMMARY -
        /LENGTH = 66 -
        /OUTPUT = RDB.REPORT -
        /STATISTICS = ALL -
        /TITLE = "Rdb Reports" -
        /OPTIONS
```

(continued on next page)

Example 2–7 (Cont.) Using Reporting Options to Generate a Customized Report

```
EVENT REQUEST_ACTUAL -
        /FACILITY = RDBVMS -
        /ITEMS = (CLIENT_PC, CPU) -
        /GROUP_BY = (CLIENT_PC) -
        /STATISTICS = (MINIMUM, MEAN) -
        /SUBTITLE = "Rdb Request Actual Summary Report"
RESTRICTION STREAM_ID 1
EVENT TRANSACTION -
        /FACILITY = RDBVMS -
        /ITEMS = (CLIENT_PC, PAGEFAULTS, PAGEFAULT_IO) -
        /TYPE = DETAIL -
        /SUBTITLE = "Rdb Transaction Detail Report"
EVENT REQUEST_ACTUAL -
        /FACILITY = RDBVMS -
        /GROUP_BY = (CLIENT_PC) -
        /TYPE = FREQUENCY -
        /INTERVAL = SECONDS -
        /SUBTITLE = "Rdb Request Actual Frequency Report"
RESTRICTION NODE RDB4ME, RDB4U
RESTRICTION COLLECTION RDB_COLL
RESTRICTION EPID 2A8002DF,2A8002C1
RESTRICTION IMAGE PAYROLL, INVENTORY
```

Data from the MY_DATABASE formatted database is used for this report. The initial REPORT command modifies several of the default qualifier values, such as the length of each page and the date before which all data should be ignored. The OPTIONS qualifier allows you to specify event and item qualifiers to either override the main qualifiers or provide additional restrictions to the report.

Based on the first part of the REPORT command, all subreports, unless otherwise specified, will contain data from the data collection file with a timestamp date greater than or equal to January 1, 1989. The report is 80 columns wide, and each page is 66 lines long. The subreports are Summary reports with all possible statistics displayed. The report is written to a file in your current directory called RDB.REPORT.

The first subreport in Example 2–7 is based on the REQUEST_ACTUAL event. The facility, a required qualifier, is RDBVMS. Of all the possible items, only the CLIENT_PC and CPU are displayed. The report is grouped by the CLIENT_PC item. This means that a subset of statistics is displayed for each different CLIENT_PC value. The CLIENT_PC is the location in the image of the actual Oracle Rdb query. Only minimum and mean statistical values are displayed for each item. A subtitle is added to the first page of the subreport. The RESTRICTION option specifies that only events with a STREAM_ID item value of 1 should be included.

The second subreport in Example 2–7 is based on the TRANSACTION event. Again, the facility, RDBVMS, is a required qualifier. The report type is overridden to provide a Detail report for this event. Only three items, CLIENT_PC, PAGEFAULTS, and PAGEFAULT_IO, are displayed for the event.

The third subreport in Example 2–7 is a Frequency report based on the REQUEST_ACTUAL event. As in the first subreport, the data is grouped by the CLIENT_PC item. The interval is broken down into seconds, so a count is displayed for each second during which at least one event occurrence was recorded. Several restrictions are specified for this report. Only data collected by the RDB_COLL collection from nodes RDB4ME and RDB4U, from processes 2A8002DF and 2A8002C1 that were running either the PAYROLL or INVENTORY programs, is displayed.

2.5.4 Improving Report Performance

If there are many occurrences of data, and you have interactive SQL on your system, you can improve report generation performance if you add an index to the REQUEST_ACTUAL table. Because the report is restricted on STREAM_ID, and grouped by CLIENT_PC, the index should be on the restricted items in the order they are entered and then on the grouped items in the order they are entered. The SQL syntax to do this is:

```
SQL> CREATE INDEX MY_INDEX ON
SQL> EPC$1_241_REQUEST_ACTUAL (STREAM_ID, CLIENT_PC);
SQL> COMMIT;
```

When you add an index it causes a delay in merging another data collection file into this formatted database. The index will not help if you want to use GROUP_BY with a different set of items. You can drop the index before you merge the databases by entering the following statements:

```
SQL> DROP INDEX MY_INDEX;
SQL> COMMIT;
```

Refer to the Oracle Trace documentation for a layout of the formatted database.

If you have interactive SQL on your system, you can look in the EPC$IMAGE table to see which images collected the data in the database. To do this enter the following SQL query:

```
SQL> SELECT IMAGE NAME FROM EPC$IMAGE.
```

See Appendix B in this manual and the Oracle Trace documentation for a description of the formatted database. ♦

3

Analyzing Performance Factors

After you put your database into production, you may find that users are experiencing changes in the time it takes to complete a task. This may mean your database is well-designed and is being used by many people; it may also mean that the performance of your database is becoming a problem. This chapter discusses the following performance factors:

- Design considerations
- Reducing disk I/O operations
- After-image journaling
- CPU utilization
- Database integrity considerations
- Constraint optimizations
- Locking
- Index retrieval
- Inserting records efficiently

Performance problems with a database are the result of many factors, including the structure of your database and the programs that use it. Database performance problems are often not apparent at first. Performance often degrades with time and increased use. Performance problems can also be the result of a poorly designed database, inefficient programming, or improperly set Oracle Rdb or operating system parameters.

3.1 Database Design Considerations

To improve database performance, you should thoroughly understand your data. Table 3–1 presents questions about data, facts derived from that data, and parameters that affect the physical design of a database. The more you know about your data, the more precisely you can estimate values for these parameters and others described in this chapter. Formulating an optimized set of estimates for these parameters enables Oracle Rdb to give your application the best performance tuning gains possible, given the nature of your application and your goals and objectives. In this way, trade-offs are matched against what is important for your application to run successfully.

The first few questions in Table 3–1 relate to logical design. Your loaded data model should be complete through third normal form showing the Entity-Relationship (E-R) mappings, pathways for all important transactions, and transaction volumes. See the *Oracle Rdb7 Guide to Database Design and Definition* for more information on logical design. The remaining questions in Table 3–1 relate to a subset of physical design parameters.

Table 3–1 How Well Do You Know Your Data?

Data Questions	Data Facts	Database Parameters Affected
Is your E-R map complete?	Entities and relationships	Index definitions, storage area definitions
Is your data organized to third normal form?	Normalized data, row size	Record definitions
Have you completed transaction analysis for all major transactions?	Database usage—read/update	Query optimization, space allocation, locking
Have you completed volume analysis?	Cardinality, growth	Space allocation, extent options
How many tables are there?	Database complexity	Query optimization
Will any tables be grouped together?	Grouped tables	Mixed page format, page size, clustering, number of buffers, buffer size
What is the row size of each table?	Record size	Space allocation, page size, fragmentation

(continued on next page)

Table 3–1 (Cont.) How Well Do You Know Your Data?

Data Questions	Data Facts	Database Parameters Affected
What is the rate of growth of each table?	Growth	Space allocation, page size, extent options, fragmentation
How are your keys defined for each table?	Primary, foreign	Index key size, index node size, fill factor, B-tree level, page size, I/O, locking
What is the size of each sorted index key?	Index key size	Index node size, fill factor, B-tree level, page size, I/O, locking
What is the number and size of index records for sorted indexes?	Cardinality, record size	Index key size, index node size, fill factor, B-tree level, page size, I/O, locking
What is the number and size of index records for hashed indexes?	Hash bucket	Index key size, index node size, page size, I/O
Which sorted indexes allow duplicates?	Parent-child relationships, duplicates allowed	Page size, B-tree levels, I/O, locking
Which hashed indexes allow duplicates?	Parent-child relationships, duplicates allowed	Page size, duplicate node records, I/O, buffer size, number of buffers, buffer size, SPAM thresholds and intervals
How many duplicate sorted index records are there?	Many duplicate records	Potential for deep B-tree, page size, I/O, locking
How many duplicate hashed index records are there?	Many duplicate records	Potential for more duplicate node records, page size, page overflow, I/O
Which indexes are unique?	Many unique indexes	Potential for shallow B-tree for sorted indexes, no duplicate node records, minimize I/O
What is the volume ratio of readers to writers on a per transaction basis?	50/50	Locking, snapshots, enabled immediate/deferred

(continued on next page)

Table 3–1 (Cont.) How Well Do You Know Your Data?

Data Questions	Data Facts	Database Parameters Affected
How many users will access the database?	Multiuser	Number of users, number of buffers, general performance
OpenVMS VAX OpenVMS Alpha How many VMScluster nodes are there? ♦	Multiuser	General performance, backup and recovery
Which tables have little growth?	Some	Compress rows
Which transactions are update (read/write) transactions?	Some	Noncompressed rows; locking, index node fill factor, fragmentation, PLACEMENT VIA INDEX clause
Which transactions are query transactions?	Some	Snapshots, full index nodes
Who needs access to what tables in what queries?	Limited access	Protection
Which queries are exact match queries?	Some	Mixed storage areas, large versus small tables, hashed indexes, sorted indexes, PLACEMENT VIA INDEX clause
Which queries are range lookup queries?	Some	Uniform storage areas, B-tree indexes
Which queries are both exact match and range lookup?	Some	Mixed and uniform storage areas, hashed and sorted indexes
Which transactions are short?	Some	Fast recovery
Which transactions are long?	Some	Slow recovery
Can transactions be optimized for recovery?	Some	Recovery time, downtime
Which transactions have high volume?	Some	Targets for optimization

You can ask many questions about your database. Their importance depends in part on what you are trying to optimize; whether you are prototyping a physical design for the first time or improving upon an existing design; your hardware configuration; the nature of your application; the types of queries on your database—that is, reads compared to updates and their approximate ratio—and many other considerations.

Some database parameters are used to calculate other database parameters; for example:

- Estimates of row size for each table determine in part the initial page size; the longest row on a mixed page format determines in part the page size.
- The total number of table rows (or cardinality) and the planned growth of the database over a time period determine the initial space allocation.
- Both page size and space allocation are used to determine SPAM intervals.
- In storage areas with uniform page format, SPAM intervals are preset based on page size; in storage areas with mixed page format, SPAM intervals vary within a minimum and maximum number of pages based on page size and space allocation. See the *Oracle Rdb7 Guide to Database Maintenance* for the formulas used to calculate SPAM intervals for both uniform and mixed page format storage areas.
- Smaller SPAM intervals can reduce SPAM page locking problems at the expense of the increased number of disk I/O operations required to locate free space for update-intensive environments, and vice versa.
- SPAM threshold values are based on size and frequency of the rows being stored relative to page size.
- Fragmentation can result from underestimating page sizes and growth of database tables.

Table 3–2 shows the interrelated nature of a few of these factors and database parameters. Incorrectly estimated parameter values, compounded by these types of interrelationships and coupled with incomplete knowledge of your data, can lead to performance problems as you bring your application on line for the first time or as your application matures.

Problems arising from miscalculated parameter values will eventually force you to respecify values that better fit your database needs using an SQL ALTER DATABASE statement or the SQL EXPORT and IMPORT statements.

Table 3–2 Interrelated Database Performance Parameters

Basic Parameter	Related Parameter	Affected by Miscalculation	Possible Problems
Record size	Page size, page format, SPAM thresholds	Fragmentation, full page, or wasted space	Too small page: get fragments; full pages: get best space usage and retrieval efficiency, but not storage efficiency; too large page: waste space.
		Record displacement	Record displacement can result from poor record sizing.
Number of records	Space allocation	Extents; correct size; wasted disk space	Small space: too many extents; moderate space: no or few extents; large space: waste disk space.
Page size	Buffer size	Buffer pool	Too small: more I/O; too large: waste buffer space and can cause page faulting in memory-poor environments; type of retrievals are affected—clustered records versus scattered records.
Page format	Uniform storage area	Constant SPAM interval based on page size, SPAM thresholds are dynamically calculated for each individual table or can be set for logical area, sorted indexes, page size, B-tree levels, index node locking with update	SPAM interval of 1089 pages for 2-block page, SPAM thresholds are dynamic by default, but can be set for individual logical areas (0 to 100%).

(continued on next page)

Table 3–2 (Cont.) Interrelated Database Performance Parameters

Basic Parameter	Related Parameter	Affected by Miscalculation	Possible Problems
Page format	Mixed storage area	Variable SPAM interval based on page size, variable SPAM thresholds, hashed indexes and sorted indexes, with hashed indexes (no) index node locking with update transactions that can occur with sorted indexes, page size	SPAM interval between 216 and 4008 pages for 2-block page; SPAM thresholds are 0 to 100%.
Global buffers enabled	NUMBER IS, USER LIMIT, and NUMBER OF BUFFERS	Buffer pool	If the buffer pool is too large, the operating system must perform more virtual paging. If the buffer pool is too small, Oracle Rdb has to perform more database I/O.
Row caching enabled	Row cache size	Wasted disk space	Row cache entries that are too large result in wasted disk space. Row cache entries that are too small prevent records from being stored in the cache.
Partition boundaries	Storage map statements—STORE USING clause with the WITH LIMIT OF clause	Partitioned record storage	Incorrect WITH LIMIT OF values or incorrect syntax for the column data type may result in all records being stored in the first partition; use an ALTER STORAGE MAP statement with the REORGANIZE option to readjust record storage.

This partial list of interrelated database parameters can guide you in your planning. It emphasizes the importance of properly estimating these database parameters when you implement your physical design or tune your application for optimal performance.

3.2 Disk I/O

This section describes how to gather information on disk input and output (I/O) factors that affect database performance and provides suggestions for reducing I/O to improve performance. Chapter 8 also provides information on examining I/O resources, balancing I/O loads, and reducing disk I/O.

3.2.1 Gathering Disk I/O Information

Sections 3.2.1.1 through 3.2.1.6 describe some of the Performance Monitor screens you can use to analyze disk I/O operations. For additional information, including I/O decision trees, see Section 8.1. For general information on how to invoke the Performance Monitor, refer to Section 2.2.

3.2.1.1 Performance Monitor Summary IO Statistics Screen

The Performance Monitor Summary IO Statistics screen summarizes database I/O activity and indicates transaction and verb execution rates. The following is an example of a Summary IO Statistics screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:04:54
Rate: 3.00 Seconds          Summary IO Statistics         Elapsed: 00:46:59.38
Page: 1 of 1            SQL_DISK1:[USER]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------

statistic.........      rate.per.second.........     total...     average..
name..............      max.     cur.     avg...     count...     per.trans

transactions               0        0        0.0            2           1.0
verb successes             2        0        1.7          250         125.0
verb failures              0        0        0.2           32          16.0

synch data reads           0        0        0.4           66          33.0
synch data writes          0        0        0.0            0           0.0
asynch data reads          0        0        0.4           66          33.0
asynch data writes         0        0        0.0            0           0.0
RUJ file reads             0        0        0.0            0           0.0
RUJ file writes            0        0        0.0            0           0.0
AIJ file reads             0        0        0.0            0           0.0
AIJ file writes            0        0        0.0            0           0.0
ACE file reads             0        0        0.0            0           0.0
ACE file writes            0        0        0.0            0           0.0
root file reads            0        0        0.1           20          10.0
root file writes           0        0        0.0            5           2.5

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in the Summary IO Statistics screen, see the Performance Monitor help.

3.2.1.2 Performance Monitor IO Stall Time Screen

The Performance Monitor IO Stall Time screen shows a summary of I/O stall activity. All times are displayed in hundredths of a second. You access the IO Stall Time screen from the IO Statistics submenu. The following example shows the IO Stall Time screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:08:26
Rate: 3.00 Seconds        IO Stall Time (seconds x100)      Elapsed: 00:50:31.47
Page: 1 of 1    KODD$:[R_ANDERSON.WORK.KUTDIS]MF_PERSONNEL.RDB;1    Mode: Online
--------------------------------------------------------------------------------
statistic..........    rate.per.second............ total....... average......
name...............    max..... cur..... avg....... count....... per.trans....

root read time                0        0       0.2          82           0.0
root write time             180        0       0.2          71           0.0

data read time                4        0       3.0        1214           0.6
data write time             100        0       0.6         239           0.1
data extend time              0        0       0.0           0           0.0

RUJ read time                 0        0       0.0           0           0.0
RUJ write time              300        0       0.3         110           0.1
RUJ extend time               0        0       0.4         141           0.1

AIJ read time                 0        0       0.1          52           0.0
AIJ write time             1100        0       8.8        3504           1.7
AIJ hiber time             1100        0       8.9        3569           1.7
AIJ extend time               0        0       0.0           0           0.0
Database bind time            0        0       0.0          21           0.0
--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in the screen, see the Performance Monitor help.

3.2.1.3 Performance Monitor Stall Messages Screen

The Performance Monitor Stall Messages screen shows a summary of database users’ stall activity. A user stalls whenever Oracle Rdb issues a system service on behalf of the user’s process. For example, a stall occurs while a user waits for a lock or for completion of a physical disk read or write.

By default, the Stall Messages screen shows all stalls, including those of millisecond duration. When a high-performance, high-volume online transaction processing (OLTP) application issues a large number of I/Os on a high-speed disk device, a database administrator (DBA) may find it impossible to differentiate between the many short millisecond stalls of the OLTP application and the longer, more important stalls that may be encountered by other applications using the database.

By typing A to select the Alarm option from the Stall Messages horizontal menu, you can specify a duration, in seconds, that a process must stall before it appears on the Stall Messages screen. For example, if you specify an alarm interval of 5 seconds, then only stalls of 5 seconds or longer duration will appear on the Stall Messages screen. If you specify a value of 0 as the alarm interval, the default, all stalls will appear on the Stall Messages screen.

By entering B once to select the Bell option from the horizontal menu, you can activate the alarm bell option and the option will be highlighted. Entering B again will deactivate the alarm bell and the option will not be highlighted.

The alarm bell, even if activated, will be rung only if the alarm option has also been activated.

When both the alarm and the alarm bell are activated, the alarm bell will be sounded once per screen refresh (specified by the Set_rate option) if there are any displayed stalls. The alarm is deactivated by setting the alarm interval to zero (0) seconds.

Stall messages are not saved in the binary statistics file created by the Output qualifier. Therefore, this screen is not available when you execute the RMU Show Statistics command in replay mode (with the Input qualifier).

You access the Stall Messages screen from the Process Information submenu.

The following example shows a Stall Messages screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:34:19
Rate: 3.00 Seconds               Stall Messages           Elapsed: 00:02:50.40
Page: 1 of 1     KODD$:[R_ANDERSON.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID Since......   Stall.reason............................ Lock.ID.
2100312E:1 13:16:00.03 - writing pages back to database
21002CAC:1 13:16:00.19 - reading pages 1:1197 to 1:1199
21002B30:1 13:16:00.17 - writing ROOT file
210030B2:1 13:16:00.18 - reading pages 1:2118 to 1:2120
210029B3:1 13:16:00.16 - writing pages back to database
21002AB4:1 13:16:00.11 - reading pages 1:4446 to 1:4448
21002638:1 13:16:00.02 - reading pages 1:4503 to 1:4505

--------------------------------------------------------------------------------
Alarm Bell Config Exit Filter Help LockID Menu >next_page <prev_page Set_rate Wr
```

This screen lists database user processes and describes the most recent stalls executed by users on the node from which the Performance Monitor was invoked. Because the stall messages are sampled only at the screen update interval, most stalls are missed. If the same stall message for a process persists, it could indicate a problem. The display also shows when a process is writing a bugcheck dump; a bugcheck dump file name longer than 53 characters is truncated.

The Stall Messages screen shows only processes that are currently stalling. Once a process finishes stalling, it disappears from the screen. Processes that are still stalling ripple up to the top of the screen. This means that the longest stalling processes appear at the top of the screen. Newer stalls are added to the bottom of the screen. Therefore, all users on the same node share the same stall display lines, and only the actively stalling processes show up on the stall screen. This allows you to monitor a relatively large number of stalling processes.

A database with no stalls has a blank stall display.

You can force frequent screen updates by using a negative number for the Time qualifier in the RMU Show Statistics command. For example, the qualifier Time=–10 refreshes the screen every 10/100 (1/10) of a second. Note that you use a lot of system resources, particularly on the smaller CPU machines, when you specify this time interval.

If more stalls are in progress than can fit on your screen, some current stalls might not be displayed. Oracle RMU attempts to place as many current stall messages on the screen as possible.

You can redirect stall messages to a file by using the Stall_Log qualifier to the RMU Show Statistics command and specifying a file name. You can also start and stop the logging of stall messages to a file from within the Peformance Monitor. Invoke the tools facility by entering the exclamation point (!), and select the Start/Stop stall logging option.

Note that message “Waiting for !AD (!AC)” displays whenever a process requests a lock “with wait” and another process is holding the lock in an incompatible mode. This message may indicate a database hot spot and should be investigated using the RMU Show Locks command. The format string “!AD” identifies the lock type (that is, storage area, page, MEMBIT, etc.) and the string “!AC” identifies the requested lock mode (PR, CR, EX, etc.).

The following list contains information on “Waiting for” messages:

- Waiting for record or page

  The “Waiting for record” and “Waiting for page” messages display a process ID, the time, and the dbkey for a record or a page.

  The dbkeys in “Waiting for record” messages are logical dbkeys. For example:

  ```
  2040037A:2   16:13:36.78  waiting for record 1:0:-4 (CR)
  202003A4:5   16:25:18.51  waiting for record 91:155:-1 (CW)
  ```

In the first line of the example, 2040037A:2 is the process ID, and 16:13:36.78 is the time. The “XX:YY:ZZ” format represents the dbkey, and you can usually interpret it as “logical area number:page number:line number.” However, only positive numbers represent the line number. When a negative number appears, it refers to the record ALG (adjustable lock granularity) locking level. By default, there are three page locking levels and the negative numbers are interpreted as follows:

–4 indicates the complete logical area
–3 normally indicates 1000 database pages range
–2 normally indicates 100 database pages range
–1 normally indicates 10 database pages range

For example, in the second line of the example, the dbkey occurs in logical area 91 in a range of 10 database pages, one of which is page 155.

When a dbkey provides a logical area number and you want to get the physical area name for that logical area, follow these steps:

1. Issue the RMU Dump command with the Lareas qualifier. For example:

   ```
   $ RMU/DUMP/LAREAS=RDB$AIP database-name
   $ rmu -dump -larea=RDB\$AIP db-name
   ```

2. Search the resulting dump for the logical area with that number.

3. Note the corresponding physical area number.

4. Issue the RMU Dump Header command. For example:

   ```
   $ RMU/DUMP/HEADER database-name
   $ rmu -dump -header db-name
   ```

   Look up the physical area number in the output of the RMU Dump Header command to find the name of the physical area.

You can also look up columns RDB$STORAGE_ID or RDB$INDEX_ID in system tables RDB$RELATIONS, RDB$INDICES, and RDBVMS$STORAGE_MAP_AREAS to identify the Oracle Rdb entity (table or index) that the dbkey represents.

The page number field in the dbkey is the number of the page in the corresponding physical area; the line number is the number of the record on that page.

The dbkeys in “Waiting for page” messages are physical dbkeys, for example:

```
202004AA:1   16:14:20.15  waiting for page 1:727 (PW)
```

This dbkey format is interpreted as “physical area number:page number.”

When a dbkey provides a physical area number and you want to get the physical area name for the area, issue the RMU Dump Header command. Then look up the physical area number in the command output to find the name of the physical area.

You can also get a conversion table by issuing the RMU Analyze command. For example:

```
$ RMU/ANALYZE/LAREAS/OPTION=DEBUG/END=1/OUTPUT=LAREA.LIS database-name
$ rmu -analyze -lareas -option=debug -end=1 -output=larea.lis db-name
```

This command produces a printable file containing the names and numbers of all the logical areas and physical areas for a database.

CR, CW, and PW in the previous examples are requested lock modes of Concurrent Read, Concurrent Write, and Protected Write. For more information on lock modes, see Table 3–8.

- Waiting for dbkey scope

  This message displays when a database user who attached using the DBKEY SCOPE IS TRANSACTION clause has a read/write transaction in progress (giving the user the database key scope lock in CW mode), and a second user who specifies the DBKEY SCOPE IS ATTACH clause (which would give the user the database key scope lock in PR mode) tries to attach. In this situation, the second user’s process stalls until the first user’s transaction completes.

  You can specify the database key scope at run time using the DBKEY SCOPE IS clause of the SQL ATTACH statement. If the DBKEY SCOPE IS clause is used with the SQL CREATE DATABASE or SQL IMPORT statements, the setting is in effect only for the duration of the session of the user who issued the statement; the setting does not become a database root file parameter.

- Waiting for MEMBIT lock

  For each Oracle Rdb database, a membership data structure is maintained. The membership data structure keeps track of the nodes that are accessing the database at any given time. The membership data structure for a database is updated when the first user process from a node attaches to the database and when the last user process from a node detaches from the database.

  The “Waiting for MEMBIT lock” message means that a process is stalled because the database’s membership data structure is in the process of being updated.

- Waiting for snapshot cursor

This message displays when a process tries to start a read-only transaction when snapshots are deferred, there is no current read-only transaction, and a read/write transaction is active.

Waiting for snapshot cursor is a normal state if snapshots are deferred. The waiting will end when all read/write transactions started before the first read-only transaction have finished.

For information about each of the fields shown in the screen, see the Performance Monitor help.

3.2.1.4 Performance Monitor DBKEY Information Screen

The Performance Monitor DBKEY Information screen helps you identify collisions on hot pages.

You access the DBKEY Information screen from the Process Information submenu. The following example shows a DBKEY Information screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor  4-JUN-1996 16:24:52
Rate: 3.00 Seconds            DBKEY Information           Elapsed: 00:03:24.45
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID  Data.Page..   Snap.Page..   Spam.Page..   AIP.Page..   ABM.Page..
7E80F841:2  1:648         14:27         5:1           1:483
7E8058F8:1  5:317         14:22         5:218         1:483
7E8052F6:1  1:648         14:20         5:218         1:483
7E80D440:1  5:65          14:18         5:1           1:483






--------------------------------------------------------------------------------
Exit Help Menu >next_page <prev_page Set_rate Write Zoom !
```

This example shows that data page 1:648 and SPAM pages 5:1 and 5:218 are commonly accessed.

The DBKEY Information screen displays, for each process attached to the database on the node, the last retrieved DBKEY for each of the following page categories: data page, snapshot page, SPAM (space area management) page, AIP (area inventory page) page and ABM (area bitmap) page.

If the attached process provides line number information as part of the data page retrieval information, the line number is displayed. Otherwise, only the area and page number are displayed. Only the area and page number are displayed for the snapshot, SPAM, AIP, and ABM pages.

3.2.1.5 Performance Monitor Active User Stall Messages Screen

The Performance Monitor Active User Stall Messages screen helps you find current stalls that represent potential deadlocks, which become database hot spots. The screen also helps you determine what stalls were last encountered by any user.

Active user stall messages are not saved in the binary statistics file created by the Output qualifier. Therefore, this screen is not available when you execute the RMU Show Statistics command in replay mode (with the Input qualifier).

You access the Active User Stall Messages screen from the Process Information submenu.

The following example shows an Active User Stall Messages screen:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:35:32
Rate: 3.00 Seconds         Active User Stall Messages       Elapsed: 00:04:02.79
Page: 1 of 1         SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
Process.ID Since......   Stall.reason............................. Lock.ID.
2100312E:1     13:16:00.03 - writing pages back to database
21002CAC:1     13:16:00.19 - reading pages 1:1197 to 1:1199
21002B30:1     13:16:00.17 - writing ROOT file
210030B2:1     13:16:00.18 - reading pages 1:2118 to 1:2120
210029B3:1     13:16:00.16 - writing pages back to database
21002AB4:1     13:16:00.11 - reading pages 1:4446 to 1:4448
21002638:1     13:16:00.02 - reading pages 1:4503 to 1:4505
210032B6:1     13:16:00.05 - Bugcheck: DISK1:[RDB]RDSBUGCHK.DMP;1

--------------------------------------------------------------------------------
Config Exit Help LockID Menu >next_page <prev_page Set_rate Write Zoom !
```

The Stall Messages screen and the Active User Stall Messages screen have the same format. However, the Active User Stall Messages screen provides information on currently stalled processes and on processes that were stalled but are no longer stalled. In contrast, the Stall Messages screen provides information only on currently stalled processes. Like the Stall Messages screen, the Active User Stall Messages screen shows when a process writes a bugcheck dump; a bugcheck dump file name longer than 53 characters is truncated.

You can force frequent screen updates by using a negative number for the Time qualifier in the RMU Show Statistics command. For example, the qualifier Time=–10 refreshes the screen every 10/100 (1/10) of a second. Note that you use a lot of system resources, particularly on the smaller CPU machines, when you specify this time interval.

If more stalls are in progress than can fit on your screen, some current stalls might not be displayed. Oracle RMU attempts to place as many active stall messages on the screen as possible.

You can redirect stall messages to a file by using the Stall_Log qualifier to the RMU Show Statistics command and specifying a file name. You can also start and stop the logging of stall messages to a file from within the Peformance Monitor. Invoke the tools facility by entering the exclamation point (!), and select the Start/Stop stall logging option.

For information about each field in the screen, see the Performance Monitor help.

The Active User Stall Messages screen has several advantages over the Stall Messages screen. The advantages are:

- The location of a process remains static; because it is fixed on a given page, it is always easy to locate.
- The process' last stall message (and lock ID, if applicable) are displayed, even if the process is not currently stalled; this is useful for identifying possible hot spots.

However, the Active User Stall Messages screen has the following disadvantages:

- It is difficult (but possible) to isolate the source of a potential deadlock or a long-duration stall; the Stall Messages screen is more useful for this.
- It is difficult (but possible) to isolate the set of currently stalled processes from the complete set of processes doing normal database accesses.

Table 3–3 shows a side-by-side comparison of the two stall messages screens.

Table 3–3 Comparison of the Stall Messages Screen and the Active User Stall Messages Screen

Category	Stall Messages	Active User Stall Messages
Processes displayed?	Only currently stalled processes on the current node are displayed.	All processes attached to the database on the current node are displayed.
Process location?	Dynamic—position reflects duration of stall relative to other processes.	Static—process remains fixed in same location until it detaches from the database.
Display sequence?	Processes are displayed in descending stall-duration sequence.	Processes are displayed in a fixed but arbitrary sequence.

(continued on next page)

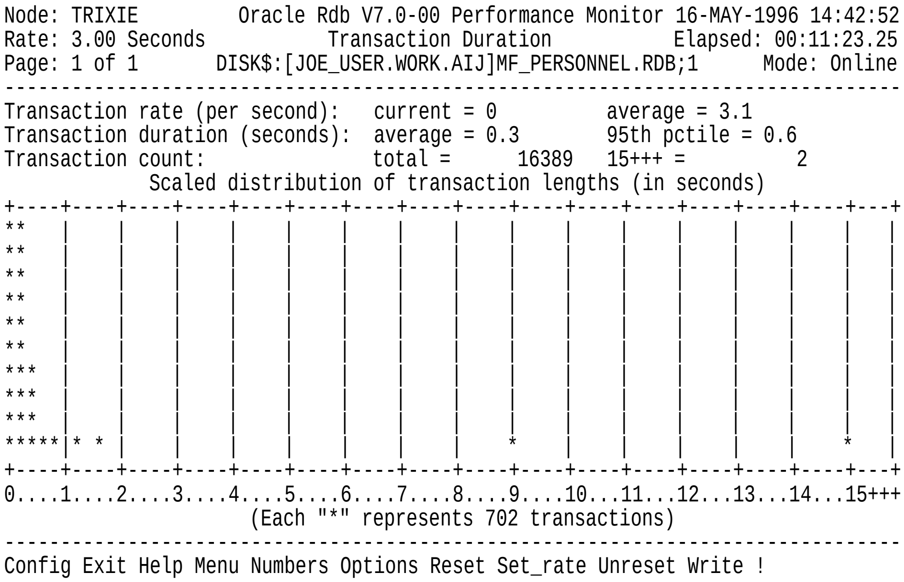
Table 3–3 (Cont.) Comparison of the Stall Messages Screen and the Active User Stall Messages Screen

Category	Stall Messages	Active User Stall Messages
Indication of current stall?	Process stall text is displayed.	Process stall text starting time is displayed.
Indication of completed stall?	Process stall text is not displayed.	Process stall text starting time is erased (message text remains).
Duration of display?	Stall message is displayed only if stall is current.	Last stall message remains displayed until the process stalls again.

3.2.1.6 Performance Monitor Transaction Duration Screen

The Performance Monitor Transaction Duration screen shows the real-time transaction processing performance of the database. You can view the Transaction Duration screen either graphically or numerically. The default display presents a graph.

The graph version of the Transaction Duration screen provides summary information about the approximate number of transactions that completed within 0 to 1 seconds, the approximate number of transactions that completed within 1 to 2 seconds, and so on through the approximate number of transactions that completed in 15 or more seconds. The following example shows the graph version of the Transaction Duration screen:



```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:42:52
Rate: 3.00 Seconds           Transaction Duration           Elapsed: 00:11:23.25
Page: 1 of 1       DISK$:[JOE_USER.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Transaction rate (per second):   current = 0          average = 3.1
Transaction duration (seconds):  average = 0.3        95th pctile = 0.6
Transaction count:               total =      16389   15+++ =          2
             Scaled distribution of transaction lengths (in seconds)
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+---+
**   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
**   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
**   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
**   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
**   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
**   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
***  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
***  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
***  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
*****|* * |    |    |    |    |    |    |    *    |    |    |    |    |    *   |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+---+
0....1....2....3....4....5....6....7....8....9....10...11...12...13...14...15+++
                      (Each "*" represents 702 transactions)
--------------------------------------------------------------------------------
Config Exit Help Menu Numbers Options Reset Set_rate Unreset Write !
```

When you use the graph version of the Transaction Duration screen, each asterisk (*) indicates some number of transactions between 1 and n, where n is the number of transactions specified at the bottom of the screen. In this example, for instance, the single asterisk in the region between 9 and 10 seconds indicates that the number of transactions taking between 9 and 10 seconds to complete could be any value between 1 and 702.

The duration of each transaction is measured from the first SQL SET TRANSACTION statement to the completion of the COMMIT or ROLLBACK statement. In addition to displaying average transaction duration, this screen shows a graph of the transaction durations.

As each transaction completes, it is added to the cumulative histogram display. The 95 percentile response time is continuously updated to show the time in which 95 percent or more of the transactions complete. The display gives an indication of subjective system response time.

The numeric version of the Transaction Duration screen provides the exact number of transactions in each of the 16 time categories. If you know the exact number of long-running transactions, you can more precisely determine how significant these transactions are in the overall performance of an application.

From the graph version of the Transaction Duration screen, type N to select the Numbers option from the horizontal menu, which changes the Transaction Duration screen from the graph version to the numeric version. The following example is the numeric version of the Transaction Duration screen shown in the previous example:

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:43:50
Rate: 1.00 Second            Transaction Duration          Elapsed: 01:27:26.56
Page: 1 of 1      DISK$:[JOE_USER.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Total transaction count:      16389
Duration   Tx.Count:   %  #Complete:   % #Incomplete:   %
 0-< 1:       16379  99%      16379  99%          10   1% <- average = 0.3
 1-< 2:           7   0%      16386  99%           3   1%
 2-< 3:           0   0%      16386  99%           3   1%
 3-< 4:           0   0%      16386  99%           3   1%
 4-< 5:           0   0%      16386  99%           3   1%
 5-< 6:           0   0%      16386  99%           3   1%
 6-< 7:           0   0%      16386  99%           3   1%
 7-< 8:           0   0%      16386  99%           3   1%
 8-< 9:           0   0%      16386  99%           3   1%
 9-<10:           1   0%      16387  99%           2   1%
10-<11:           0   0%      16387  99%           2   1%
11-<12:           0   0%      16387  99%           2   1%
12-<13:           0   0%      16387  99%           2   1%
13-<14:           0   0%      16387  99%           2   1%
14-<15:           0   0%      16387  99%           2   1%
 15+++:           2   0%      16389 100%           0   0%
--------------------------------------------------------------------------------
Config Exit Graph Help Menu Options Reset Set_rate Unreset Write !
```

In addition to displaying the exact number of transactions in each time category, the numeric version shows the number of transactions that completed and the number of transactions that did not complete within each time category.

For information about each field in the screen, see the Performance Monitor help.

If you collect statistics in an output file using the Output qualifier for the RMU Show Statistics command, you can replay the Transaction Duration screen by using the Input qualifier.

From the numeric version of the Transaction Duration screen, type G to select the Graph option from the horizontal menu, which changes the screen from the numeric version to the graph version.

3.2.1.7 Reducing Disk I/O

Oracle Rdb provides the following features that can help reduce disk I/O:

- Global buffers

  When you enable global buffering, Oracle Rdb sets up a pool of buffers that can be shared by all database users on a node. This prevents two users from reading the same page into memory, thus saving disk I/Os. See Section 4.1.2 for information on how to use global buffers.

- Row caches

  When you enable row caching, Oracle Rdb allocates memory to store rows that are frequently accessed. The rows remain in the row cache even when the associated page has been flushed back to disk. The code path is shorter when a row is retrieved from a row cache as compared to a page buffer pool or disk. See Section 4.1.3 for information on how to use row caches.

- Fast commit processing

  When you enable fast commit processing, Oracle Rdb delays writing committed updates to disk until a user-specified threshold, called a checkpoint, is reached. All transactions since the last checkpoint are written to disk at the same time, which is more efficient than writing updates to disk after each committed transaction. Also, you save RUJ I/Os because the RUJ buffer does not get flushed to the .ruj file after each transaction. See Section 4.1.5 for information on how to use fast commit processing.

When executing queries that yield a large record stream, the optimizer may have to create an intermediate table to store the results of a subquery. Oracle Rdb stores these results in sorted order for further execution in join operations. You can use the logical name RDMS$DEBUG_FLAGS or the RDB_DEBUG_FLAGS configuration parameter to determine if the optimizer is using an intermediate table (see Section C.1 for more information).

You may be able to complete more transactions in a given time if you define either or both of two logical names or configuration parameters recognized and used by Oracle Rdb. The logical names are RDMS$BIND_WORK_FILE and RDMS$BIND_WORK_VM. The configuration parameters are RDB_BIND_WORK_FILE and RDB_BIND_WORK_VM.

OpenVMS VAX OpenVMS Alpha The logical names RDMS$BIND_WORK_FILE and RDMS$BIND_WORK_VM may be defined at a system, group, or process level on an OpenVMS system.

Because the Oracle Rdb work file on an OpenVMS system is an RMS file, you can also set the RMS multibuffer and multiblock counts by using the DCL SET RMS_DEFAULT command to specify appropriate values for the BUFFER_COUNT and BLOCK_COUNT qualifiers and improve the performance of Oracle Rdb temporary tables on OpenVMS systems. ♦

The RDMS$BIND_WORK_VM logical name and RDB_BIND_WORK_VM configuration parameter can reduce the overhead of disk I/O operations for matching operations. This logical name or configuration parameter lets you specify the amount of virtual memory (VM) in bytes that will be allocated to your process for use in matching operations. Once the allocation is exhausted, additional data values are written to a temporary file on disk (SYS$LOGIN on

OpenVMS, if RDMS$BIND_WORK_FILE is undefined). The default is 10,000 bytes. The maximum allowed value is restricted only by the amount of memory available on your system.

OpenVMS VAX OpenVMS Alpha

The following example defines the RDMS$BIND_WORK_VM logical name to be 20,000 bytes.

```
$ DEFINE RDMS$BIND_WORK_VM 20000
```

The number 20,000 is a byte count and is deducted from your process quota value, PGFLQUOTA, in SYSUAF.DAT. As the byte count increases, user processes increase paging because the I/O operations have been transferred to the paging disk. Experiment with the values that give the highest throughput for your application. See Section 8.2 and Section 4.4.3 for more information on memory management and establishing reasonable working set parameter values for each process. See the OpenVMS documentation set for guidelines on setting initial working set values for tuning automatic, working set adjustment parameters for these two general strategies:

- Tuning for rapid response times whenever the workload demands greater working set sizes in an ever-changing, timesharing environment
- Tuning for less dynamic response times that will stabilize and track moderate needs for working set growth in a production environment ♦

The logical name RDMS$BIND_WORK_FILE or the RDB_BIND_WORK_FILE configuration parameter lets you specify the disk device and directory for temporary work files.

OpenVMS VAX OpenVMS Alpha

On OpenVMS systems, the default is to create temporary files in your home directory. Oracle Rdb uses these files in matching operations. The following example defines a temporary work file through the RDMS$BIND_WORK_FILE logical name to be on WORK$DISK:[RDB.WORK]:

```
$ ! Assign the work area to another disk with read/write access:
$ DEFINE RDMS$BIND_WORK_FILE WORK$DISK:[RDB.WORK]
```

♦

Another way to reduce disk I/O operations is to control placement of .ruj files by using the logical name RDMS$RUJ or the configuration parameter RDB_RUJ.

3.2.2 Data Distribution

You might be able to improve access to data that is clustered around a few tables by distributing the data of interest across more tables, thus implementing further normalization.

Designing the conceptual or logical database should account for both referential (read-only) and application (read/write) tables. The application tables, where clustering can occur, may require certain refinements or alterations so they can support users' information requirements. If your design results in 2 or 3 out of perhaps a total of 10 tables where 90 percent of the database activity occurs, you may achieve a performance gain by one or more of the following actions:

- Some compromise in normalization
- More frequent archival of inactive rows
- A change in your indexing strategy
- Judicious use of constraints
- A stricter protection policy
- Definitions of more views

For more information on data distribution, see Section 8.1.2.3. For information on constraints and defining views, see the *Oracle Rdb7 Guide to Database Design and Definition*.

3.2.3 Data Content—Active Versus Inactive Rows

Once your database is loaded and begins to support one or several applications, the number of rows in the database may grow beyond reasonable limits or expectations. As database administrator, you should ensure that only data that supports current applications resides in the database. Keeping data that is not current (historical data), in the same database as active rows can degrade performance significantly when kept in active storage areas. This problem has several possible solutions, with the first one being one of the best alternatives:

- Make storage areas that contain stable data read-only to minimize locking conflicts.

  Storage areas that contain historical data or data that is not subject to change over long periods of time can be made read-only to reduce locking conflicts. To accomplish this, either move the stable data to new storage areas and set these to read-only, or make the entire storage area read-only.
- Unload historical data to another table on another device or to tape, depending on access frequency.

You can define an identical database structure on another device to contain historical data for a specified period. When access to that data is no longer needed, it can be deleted or loaded to a tape as required.

- Place the data into summary rows for storage in another table, or in another medium, such as microfiche or CD–ROM.

  You can produce periodic summaries of inactive rows and store this data in summary tables elsewhere on your system. In this way, you keep the information, but reduce the data storage requirements.

By using a combination of these methods, you can maintain your primary data resource at its optimum efficiency.

3.2.4 Asynchronous Prefetch of Database Pages

The Oracle Rdb asynchronous prefetch feature reduces the amount of time that a process waits for pages to be read from disk by fetching pages before a process actually requests the pages. When the asynchronous prefetch feature is enabled, Oracle Rdb examines each process and attempts to predict the process' future access patterns. When Oracle Rdb can predict a sequential scan from a process' successive page requests, it fetches the pages that it anticipates will be included in the sequential scan. This prefetching of pages (fetching a page before it has been requested) is asynchronous. That is, Oracle Rdb does not wait for the prefetched pages to be read into memory from disk; it continues with its current processing activities.

Asynchronous prefetch is enabled by default. Example 3–1 shows how to disable the asynchronous prefetch feature using SQL syntax.

Example 3–1 Disabling the Asynchronous Prefetch Feature

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ASYNC PREFETCH IS DISABLED;
```

To enable the asynchronous prefetch feature after it has been disabled, use the ASYNC PREFETCH IS ENABLED clause of the ALTER DATABASE statement.

You can also use the RDM$BIND_APF_ENABLED logical name or the RDB_BIND_APF_ENABLED configuration parameter to disable and enable the asynchronous prefetch feature for a process.

OpenVMS VAX OpenVMS Alpha

The following example shows how to disable and enable the asynchronous prefetch feature on OpenVMS:

```
$ ! Disable the asynchronous prefetch feature:
$ !
$ DEFINE RDM$BIND_APF_ENABLED 0
$ !
$ ! The RDM$BIND_APF_ENABLED logical name is translated when a
$ ! process attaches to the database.  This logical name can be
$ ! a system or process logical.
$ !
$ ! To enable the asynchronous prefetch feature after it has been
$ ! disabled for a process, set the RDM$BIND_APF_ENABLED logical name
$ ! to 1:
$ !
$ DEFINE RDM$BIND_APF_ENABLED 1
$ !
$ ! Or, deassign the logical name:
$ DEASSIGN RDM$BIND_APF_ENABLED
```

♦

Oracle Corporation recommends that you use SQL syntax instead of the logical name or configuration parameter. See the *Oracle Rdb7 SQL Reference Manual* for information about the ASYNC PREFETCH clause of the SQL CREATE DATABASE, ALTER DATABASE, or IMPORT statements.

The page on which Oracle Rdb determines a sequential scan is being requested by a process is called a **triggering page**. When Oracle Rdb discovers a triggering page, it begins prefetching pages. Oracle Rdb uses different methods of prefetching, depending on whether the triggering page is in a mixed format storage area or a uniform format storage area, as explained later in this section.

The number of buffers that Oracle Rdb can prefetch for a process is called the **depth of buffers**. Oracle Rdb determines the default depth of buffers for a process by using this formula:

$$DEFAULT\ DEPTH\ IS\ SMALLER\ VALUE\ OF\ \frac{buffers\ in\ allocate\ set}{4}\ OR\ 8$$

If, for example, a process had 100 buffers in its allocate set, the default would be either 25 buffers (100 buffers divided by 4 equals 25 buffers) or 8 buffers. Because 8 buffers is less than 25 buffers, Oracle Rdb will use 8 buffers of the process' allocate set for prefetching.

You can change the default depth by using the DEPTH IS parameter of the ASYNC PREFETCH IS ENABLED clause. Example 3–2 shows how to specify that 10 buffers of a process' allocate set be used for prefetching.

Example 3–2 Specifying the Number of Buffers to Be Prefetched

```
SQL> ALTER DATABASE FILE mf_personnel
cont> ASYNC PREFETCH IS ENABLED
cont> (DEPTH IS 10);
SQL>
```

When Oracle Rdb finds a triggering page in a mixed format storage area, it can determine the next page (the first page to prefetch) by adding 1 to the triggering page number. In a mixed format storage area, Oracle Rdb optimizes disk access by prefetching the depth of buffers for the process in a single read operation. Oracle Rdb examines the depth of buffers for the process and continues prefetching if it finds another triggering page.

When Oracle Rdb finds a triggering page in a uniform format storage area, it can determine the next page in the logical area (the first page to prefetch) from the SPAM page for the triggering page. In a uniform storage area, Oracle Rdb also prefetches the depth of buffers for the process one at a time, using a separate read operation for each buffer in the depth of buffers (in contrast to prefetching in mixed format storage areas, where all the buffers in the depth of buffers are read in a single read operation). The depth of buffers for the process is examined, and if Oracle Rdb finds another triggering page, it continues to prefetch buffers. The prefetching in a uniform storage area takes place at the SPAM page and data page levels; Oracle Rdb uses area bit map (ABM) pages to determine which SPAM pages map to which database page clumps for a specific logical area.

When Oracle Rdb begins prefetching for either a mixed format or a uniform format storage area, it examines each prefetched page one buffer at a time. Note that the asynchronous prefetch request is ignored if a page to be prefetched is found in memory.

The advantage of the asynchronous prefetch feature is that Oracle Rdb can reduce the number of times it needs to stall while waiting for a page to be read from disk. When asynchronous prefetching of pages is not enabled, Oracle Rdb has to stall for each read operation from disk. Oracle Rdb does not have to stall when it fetches a page asynchronously; therefore, each time it correctly predicts a page that a process will need before the process requests it, Oracle Rdb saves the stall time that would normally have been associated with the reading of that page. This can substantially improve performance for a wide range of applications.

The asynchronous prefetch feature is most likely to improve the performance of processes with allocate sets of 20 or more buffers.

You can use the Performance Monitor Asynchronous IO Statistics screen to get information on the asynchronous prefetch operations occurring in your database. See Section 4.1.1.5 for more information on the Asynchronous IO Statistics screen.

3.2.5 Asynchronous Batch-Write Operations

Oracle Rdb reads and writes pages while executing transactions. By default, it supports asynchronous batch-write operations, which reduce the number of stalls experienced by database processes while waiting for writes to disk to complete.

The goal of asynchronous batch-write operations is to increase database performance by making it possible for a certain number of buffers in each process' allocate set to have write operations in progress at any time without causing the process to stall. For each individual process, performance is best when asynchronous write operations are issued before all the buffers in the process' allocate set have been modified (see Section 4.1.2.2 for more information on determining the number of buffers Oracle Rdb allocates to a user process).

The following benefits are provided by the asynchronous batch-write feature:

- Reduces server stalls
- Fewer servers are required to obtain the same application throughput because each server is more efficient
- Better response time for update applications
- Faster performance by Oracle Rdb utilities that perform write operations, such as RMU Recover, RMU Load, and database recovery (DBR) processes

This feature is most likely to improve the performance of processes with allocate sets of 20 or more buffers.

Asynchronous batch-write operations are enabled by default. Example 3–3 shows how to disable asynchronous batch-write operations using SQL syntax.

Example 3–3 Disabling Asynchronous Batch-Write Operations

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ASYNC BATCH WRITES ARE DISABLED;
```

You can also use the RDM$BIND_ABW_ENABLED logical name or the RDB_BIND_ABW_ENABLED configuration parameter to disable and enable asynchronous batch-write operations for a process.

OpenVMS VAX OpenVMS Alpha

The following example shows how to use the RDM$BIND_ABW_ENABLED logical name on OpenVMS:

```
$ ! Disable asynchronous batch-write operations:
$ !
$ DEFINE RDM$BIND_ABW_ENABLED 0
$ !
$ ! The RDM$BIND_ABW_ENABLED logical name is translated when a
$ ! process attaches to the database.  This logical name can be
$ ! a system or process logical.
$ !
$ ! To enable asynchronous batch writes after they have been disabled
$ ! for a process, set the RDM$BIND_ABW_ENABLED logical name to 1:
$ !
$ DEFINE RDM$BIND_ABW_ENABLED 1
$ !
$ ! Or, deassign the logical name:
$ DEASSIGN RDM$BIND_ABW_ENABLED
```

♦

Oracle Corporation recommends that you use SQL syntax instead of the RDM$BIND_ABW_ENABLED logical name or the RDB_BIND_ABW_ENABLED configuration parameter. See the *Oracle Rdb7 SQL Reference Manual* for information about the ASYNC BATCH WRITES clause of the SQL CREATE DATABASE, ALTER DATABASE, or IMPORT statements.

You use the CLEAN BUFFER COUNT and the MAXIMUM BUFFER COUNT parameters to control asynchronous batch-write operations for a process. You use the CLEAN BUFFER COUNT parameter to specify the number of clean buffers to be maintained at the end of a process' least recently used queue of buffers for replacement. In Example 3–4, the CLEAN BUFFER COUNT is defined as 4 to specify that four clean buffers be maintained at the end of the process' least recently used queue of buffers for replacement.

Example 3–4 Specifying the Number of Clean Buffers to Be Maintained

```
SQL> ALTER DATABASE FILE mf_personnel
cont> ASYNC BATCH WRITES ARE ENABLED
cont> (CLEAN BUFFER COUNT IS 4 BUFFERS);
```

You use the MAXIMUM BUFFER COUNT parameter to specify the number of buffers a process will write asynchronously. In Example 3–5, the MAXIMUM BUFFER COUNT is defined as 6 to specify that six modified buffers be written asynchronously whenever the number of clean buffers for the process is less than the value specified by the CLEAN BUFFER COUNT value.

Example 3–5 Specifying the Number of Buffers to Be Written Asynchronously for a Process

```
SQL> ALTER DATABASE FILE mf_personnel
cont> ASYNC BATCH WRITES ARE ENABLED
cont> (MAXIMUM BUFFER COUNT IS 6 BUFFERS);
```

For a process with the CLEAN BUFFER COUNT and MAXIMUM BUFFER COUNT values specified in Example 3–4 and Example 3–5, Oracle Rdb will write a group of six modified buffers to disk asynchronously whenever the process has fewer than four clean buffers in the least recently used queue of buffers for replacement. The process does not wait for the write operation of the six modified buffers to complete; it continues processing while the asynchronous batch-write operation occurs.

You probably will need to try different CLEAN BUFFER COUNT and MAXIMUM BUFFER COUNT values for a process to determine which values provide the best performance. In general, performance is better when the value for the CLEAN BUFFER COUNT is not set too high; a high value can increase CPU usage. However, you also want to anticipate asynchronous batch-write operations and have Oracle Rdb start them while enough clean buffers remain to allow processing to continue without a stall, so do not set the value too low. The default value for the CLEAN BUFFER COUNT is 5.

The MAXIMUM BUFFER COUNT value for a process should not be set too high. If it is, the process' batch-write operations may be so large that I/Os from other processes are queued up behind the process' batch-write operations (this is especially likely if all the batch-write operations are to a single disk).

If the MAXIMUM BUFFER COUNT value for a process is too low, then the process will not benefit from disk parallelism. For example, if the MAXIMUM BUFFER COUNT value for a process is 2 and five disks are written to by the application, then the process is only writing asynchronously to a maximum of two disks with any batch-write operation. A larger MAXIMUM BUFFER COUNT value could increase the number of disks written to with each asynchronous batch-write operation. Also, if the MAXIMUM BUFFER COUNT value for a process is too low, a process reduces the benefit of disk optimizations. For example, when an application updates a large number of values, Oracle Rdb sorts these values by their page number before it writes the updated values to disk. This sorting of values by page allows the disk head to write all the updates for the same page in one operation instead of writing one updated value for a page at one time, then returning to the page later to write another updated value to it. When the MAXIMUM BUFFER COUNT value is too low, the number of values that is sorted is small; therefore, several write operations are required to write all the updates to a single page.

The correct MAXIMUM BUFFER COUNT is one that permits Oracle Rdb to always be steadily writing some buffers for the process. The default value for the MAXIMUM BUFFER COUNT is all marked buffers, so usually you would use the MAXIMUM BUFFER COUNT syntax to specify that Oracle Rdb asynchronously write fewer marked buffers for a process.

Although a process using asynchronous batch-write operations experiences fewer stalls, it will stall when Oracle Rdb needs to replace a buffer and an asynchronous batch-write operation is in progress. This situation can be remedied by increasing the CLEAN BUFFER COUNT value. A stall can also occur if a process receives a blocking AST from another process for a page when an asynchronous batch-write operation is in progress. A process will also stall when it needs to demote locks for a buffer as part of page deadlock handling, or when Oracle Rdb performs a checkpoint operation.

You can use the Performance Monitor Asynchronous IO Statistics display to get information on the asynchronous batch-write operations occurring in your database. See Section 4.1.1.5 for more information on the Asynchronous IO Statistics display.

3.3 CPU Utilization

The Performance Monitor provides a CPU Utilization screen that shows the current CPU utilization of each database process on the node as a percentage of the total processor utilization.

You select the CPU Utilization screen from the Process Information submenu. For example:

```
+------------ Select Display -----------+
|                                       |
|  A. Stall Messages                    |
|  B. Active User Stall Messages        |
|  C. Process Accounting                |
|  D. Checkpoint Information (Unsorted) |
|  E. Active User Chart                 |
|  F. CPU Utilization (Unsorted)        |
|  G. DBR Activity                      |
|  H. Monitor Log                       |
|  I. Defined Logicals                  |
|  J. Lock Timeout History              |
|  K. Lock Deadlock History             |
|  L. DBKEY Information                 |
|                                       |
+---------------------------------------+
```

Typing F selects the CPU Utilization screen. For example:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:47:12
Rate: 1.00 Second          CPU Utilization (unsorted)       Elapsed: 00:49:12.10
Page: 1 of 1     KODD$:[R_ANDERSON.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID Process.name... CPU.Util%      10  20  30  40  50  60  70  80  90 100
                                       +---+---+---+---+---+---+---+---+---+---+
2720B4D6:1 RICK2                  9%   +--*|   |   |   |   |   |   |   |   |   +
2720D279:1sRDM_ALS611_1           1%   |   |   |   |   |   |   |   |   |   |   |
2720C687:1sRDM_ABS611_1           0%   |   |   |   |   |   |   |   |   |   |   +
                                       |   |   |   |   |   |   |   |   |   |   |
---------------------------------------+---+---+---+---+---+---+---+---+---+---+
Config Exit Help Menu >next_page <prev_page Set_rate Write Zoom !
```

The CPU Utilization screen provides one line of information for each database process active on the node. If a process is attached to the database multiple times, only one line of information will be displayed.

Each line of the screen shows specific CPU utilization information associated with a database process. The process ID and name of each process is displayed, followed by the CPU utilization percentage of that process. The histogram portion of the screen show the process' utilization percentage in graph format.

Note that the displayed process CPU utilization, in addition to the database-related processing, includes the CPU utilization of all nondatabase-related activities that the process also performs. To track the exact CPU utilization of the database-specific activity, Oracle Corporation recommends you use Oracle Trace software[1].

Under the 100% column of the histogram, a plus sign (+) indicates that the process incurred 100% CPU utilization at some time during the display session. Leaving the screen and returning to it later resets this indicator.

[1] On OpenVMS systems only

By default, the processes on the screen appear in an arbitrary (unsorted) order. By selecting the Config option, you can manually configure the CPU Utilization screen. For example, typing C displays the CPU Utilization configuration menu:

```
+- Select CPU Utilization Configuration -+
|                                        |
|  A.  Unsorted                          |
|  B.  Sort by CPU Utilization           |
|                                        |
+----------------------------------------+
```

Option A displays the processes on the CPU Utilization screen in unsorted order.

Option B displays the processes on the CPU Utilization screen sorted in order of descending CPU utilization. This configuration can be difficult to view when the display rate is less than 2 seconds, because the process CPU utilizations vary at any given time. Oracle Corporation recommends the sorted display be used with a display rate of 2 seconds or more, as this provides a more linear view of the top CPU utilization processes.

The CPU Utilization screen is not saved by using the Output qualifier; therefore, it cannot be replayed using the Input qualifier.

3.4 Gathering Database Root File Information

The database root file is comprised of a series of components known as objects. Each object identifies a particular class of information about the database, such as AIJ or storage area information. Do not confuse root file objects with object-oriented objects; they are unrelated.

When the database is opened by the monitor, the objects are read from the database root file and stored in the database global section on OpenVMS and in the shared memory partition on Digital UNIX. During database physical modification, as well as during run time, objects are occasionally refreshed from the root file into the global section or shared memory partition, modified, and then written back to disk. For instance, to allocate a transaction sequence number (TSN) when a transaction is started, the SEQBLK object is fetched, modified, and written back to disk.

You can use the following Performance Monitor screens to analyze I/O operations being performed on the database root file:

- Summary Object Statistics

  The Summary Object Statistics screen provides cumulative information for all database root file objects.

- Objects (one stat type)

  The Objects (one stat type) screen allows you to display statistics for a specific object. When you select the Objects (one stat type) option from the menu, a submenu containing the root file objects is displayed. You can then select an object from the submenu, and statistics for the selected object, broken out by category, will be displayed.

- Objects (one stat field)

  The Objects (one stat field) screen allows you to display statistics for a specific collection category. When you select the Objects (one stat field) option from the menu, a submenu containing the categories is displayed. You can then select a category from the submenu, and the statistics for that category will be displayed, broken out by object type.

For information about each field in these screens, see the Performance Monitor help.

3.5 After-Image Journaling

No database activity in a production environment should occur without enabling the after-image journaling facility. If system failure occurs, you can restore your database application from the most recent full backup file, then roll forward the .aij file to capture all completed transactions that were made since the last full backup operation. In this way, you ensure that the database can be brought back to its most current state prior to the system failure, which includes rolling forward all completed transactions since the database was last backed up. The journaling strategy you choose depends on several factors listed here. Any of these factors, or all of them together, can significantly influence how well the database responds to users' requirements:

- The number of users normally accessing the database

  Whether a few users or many users routinely access the database, if a system failure occurs while a user is entering data, that information is lost. If all transactions are designed to be as short as possible, the worst case may be that only the last uncommitted active transaction is lost. User-process recovery can occur only after system recovery. The user need only enter data again from the last source document or point-of-sale transaction. Larger transaction scopes can result in unsatisfactory delay in recovery because the user must recover the process by entering the last set of uncommitted updates. Furthermore, procedures must exist that permit the user to determine the exact state of the database. This activity, too, takes valuable time.

- Transaction volume

When the database experiences a state of maximum update support for extended periods each day, it is possible that several thousand update transactions are recorded in the .aij file for that period alone. Use the RMU Backup Online command to create a backup of the database itself at regular intervals. You do not need to close the database to active users to do this kind of backup operation; however, this operation waits for a quiet point and then proceeds. You can also use the Noquiet_Point qualifier if you want the backup operation to begin when the RMU Backup command is issued, regardless of any read/write transactions in progress in the database.

Depending on transaction volume, the .aij file can grow very large, require a large amount of disk storage, and hinder database performance. In this case, use the RMU Backup After_Journal command occasionally. The RMU Backup After_Journal command directs Oracle Rdb to copy information from the .aij file to a file on a tape drive, or to an alternate disk drive. For more information on the RMU Backup After_Journal command, see the *Oracle RMU Reference Manual* and the *Oracle Rdb7 Guide to Database Maintenance*.

- The maximum acceptable time the database is allowed to be unavailable
- The dependence of other databases on your database

  Databases on other devices or nodes may require data in your database. For example, the payroll department accesses employee data from the mf_personnel database when processing payroll. If the availability of your database is interrupted because of a device failure, or needs more time to recover after system failure because the journal file is excessively large, the dependent processes must remain idle, thus causing inefficiencies in your management information system.

Oracle Rdb allows you to use either a single extensible journal or multiple fixed-size journals for after-image journaling. In most cases, Oracle Corporation recommends using multiple fixed-size journals. See the chapter on after-image journaling in the *Oracle Rdb7 Guide to Database Maintenance* for a complete discussion on the advantages and disadvantages of each type of journal.

Section 8.1.2.2 describes how to check for performance problems related to after-image journaling.

3.5.1 Using the AIJ Log Server (ALS) to Improve the Performance of After-Image Journal File Write Operations to Disk

Multiple-user databases with medium to high update activity sometimes experience after-image journal (.aij) file bottlenecks. You can eliminate .aij file bottlenecks by using a special process called the AIJ log server (ALS).

When the ALS is not enabled, user processes need to get a local lock for their own updated buffers and the updated buffers of other processes in a database's global section or shared memory partition before doing a group commit of the updated buffers to the disk on which the .aij file resides. When high update activity is occurring in the database, this lock contention can cause a decrease in performance.

To alleviate these .aij file bottlenecks, Oracle Rdb allows you to use the AIJ log server (ALS) to flush log data to the .aij file. When the ALS is used, all database user processes leave the log data in a global section or shared memory partition and the ALS flushes the log data to disk on behalf of all the user processes. Because the ALS is a dedicated process, writing to the .aij file is much faster than when the ALS is not used. Also, there is no contention between user processes for the local lock in the global section or shared memory partition when ALS is used, because the ALS is the only process that ever requests the lock.

Before you begin using the ALS process for a database, you first need to decide whether you want to use the automatic or manual startup mode for ALS. The ALTER DATABASE statement in Example 3–6 specifies the automatic ALS startup mode for the database. The Performance Monitor Journaling Information screen displays the ALS startup mode for the database.

Example 3–6 Specifying the Automatic Startup Mode for the ALS Process

```
SQL> -- Specifying the automatic startup mode for the ALS
SQL> ALTER DATABASE FILENAME database-name
cont> JOURNAL ENABLED (LOG SERVER IS AUTOMATIC);
SQL>
```

If ALS startup mode is automatic and the database open mode is automatic, the ALS starts on a node when the first process from the node attaches to the database; the ALS stops when the last process from the node detaches, when the RMU Server After_Journal Stop command is issued, or when the RMU Close command is issued. If ALS startup mode is automatic and the database open mode is manual, the ALS starts on a node when the database is explicitly opened with the RMU Open command; the ALS stops when the database is explicitly closed with the RMU Close command or when the RMU Server After_Journal Stop command is issued.

Table 3–4 shows, for databases with open modes automatic and manual, how the ALS is started and stopped when the ALS startup mode is set to automatic.

Table 3–4 Starting and Stopping the ALS When the ALS Startup Mode Is Automatic

If Database Open Mode Is:	ALS Starts:	ALS Stops:
Automatic	When the first process attaches to the database	When the last user detaches from the database, or the RMU Server After_Journal Stop or RMU Close command is issued
Manual	When the database is opened with the RMU Open command	When the database is closed with the RMU Close command or when the RMU Server After_Journal Stop command is issued

By default, the ALS startup mode is set to manual. The ALTER DATABASE statement in Example 3–7 sets the startup mode to manual.

Example 3–7 Specifying the Manual Startup Mode for the ALS Process

```
SQL> -- Specifying the manual startup mode for the ALS
SQL> ALTER DATABASE FILENAME database-name
cont> JOURNAL ENABLED (LOG SERVER IS MANUAL);
SQL>
```

When the ALS startup mode for a database is set to manual, you must use the RMU Server After_Journal Start command to start the ALS on a node, as shown in Example 3–8.

Example 3–8 Manually Starting the ALS Process on a Node

```
$ RMU/SERVER AFTER_JOURNAL START database-name
$ rmu -server after_journal start database-name
```

Note that the RMU Server After_Journal Start command will start the ALS process only if the database is open. This means that if the ALS startup mode is manual and the database open mode is automatic, the ALS will start only if at least one process is attached to the database from the node from which the RMU Server After_Journal Start command is issued. If the ALS startup mode is manual and the database open mode is manual, the ALS will start only if the database has been explicitly opened with an RMU Open command from the node from which the RMU Server After_Journal Start command is issued. In other words, if a database is closed when you issue the RMU Server After_Journal Start command, the ALS process will not be started.

When the ALS startup mode is manual, the ALS stops when the RMU Server After_Journal Stop command is issued. Example 3–9 shows how to use the RMU Server After_Journal Stop command to manually stop the ALS process for a database on a node. You can use this command even when users are accessing the database. When the RMU Server After_Journal Stop command is used to stop the ALS process, any active processes still using the database must write their own updated buffers and the updated buffers of other users to the .aij file in group commit operations. You can also stop the ALS process by issuing an RMU Close command; in this case, all the processes in the database are stopped as part of the database shutdown.

Example 3–9 Manually Stopping the ALS Process on a Node

```
$ RMU/SERVER AFTER_JOURNAL STOP database-name
$ rmu -server after_journal stop database-name
```

You can start and stop the ALS process while the Performance Monitor is executing. Invoke the tools facility by entering the exclamation point (!), and select the Start/Stop AIJ Log Server option.

When the ALS startup mode for a database is set to manual and the ALS process has been manually started for the database using the RMU Server After_Journal Start command, the ALS process remains in effect until it is manually stopped with an RMU Server After_Journal Stop command (assuming that the RMU Close command is not used to close the database). Even if all the database users on a node detach from the database, the ALS process remains in effect.

Table 3–5 shows, for databases with open modes automatic and manual, how the ALS is started and stopped when the ALS startup mode is set to manual.

Table 3–5 Starting and Stopping the ALS When the ALS Startup Mode Is Manual

If Database Open Mode Is:	ALS Starts:	ALS Stops:
Automatic	When at least one process is attached to the database and the RMU Server After_Journal Start command is issued	When the RMU Server After_Journal Stop or the RMU Close command is issued
Manual	When the database is opened with the RMU Open command and the RMU Server After_Journal Start command is issued	When the RMU Server After_Journal Stop command is issued or the database is closed with the RMU Close command

You can determine whether or not the ALS process is started for a database on the current node by using the RMU Show Users command. In Example 3–10, the process RDM_ALS701 is the ALS process. If the ALS process is not started for the database, information on the RDM_ALS701 process does not appear in the RMU Show Users display.

OpenVMS VAX ≡ OpenVMS Alpha ≡

Example 3–10 Determining When the ALS Process Is Started for a Database

```
$ RMU/SHOW USERS mf_personnel
Oracle Rdb V7.0-00 on node TRIXIE 16-MAY-1996 14:51:26.32

database $111$DUA68:[RICK]MF_PERSONNEL.RDB;1
    - after-image journal file is SQL_DISK1:[RICK.V70_AIJS]AIJ2.AIJ;1
    - 2 active database users

    - 3D01029D:1 - RICK, RICK - active user
        - image $111$DUA33:[SQL.V070.][SRC.MID]SQL$70.EXE;2

    - 3D00F921:1 - RDM_ALS701_1, RDBVMS - active user
        - image $111$DUJ0:[SYS5.SYSCOMMON.][SYSEXE]RDMALS701.EXE;49
$
```

♦

If you do not specify a database name with the RMU Show Users command, information on the users of all the active databases on the node is displayed if:

- You have the OpenVMS WORLD, SYSPRV, or BYPASS privilege on OpenVMS
- You are the dbsmgr user or the superuser on Digital UNIX

If the ALS process is started for a database, each node that has the database open will have its own ALS process for user processes accessing the database from that node.

If the ALS process stops abnormally (by some means other than an RMU Server After_Journal Stop command or an RMU Close command, or by the last process detaching from the database), it is not restarted automatically. The ALS process is recovered by the database recovery (DBR) process, but it needs to be restarted manually. The RMU Server After_Journal Start command is normally used only for databases with the ALS mode set to manual. However, if the ALS process stops abnormally, the RMU Server After_Journal Start command must be used to start the ALS process again, regardless of whether the database's ALS startup mode is manual or automatic.

OpenVMS VAX OpenVMS Alpha

ALS process bugcheck dumps are written to the RDMALSBUG.DMP file in the SYS$SYSTEM directory. ♦

3.5.2 Improving ALS Process Performance by Using an AIJ Cache on an Electronic Disk

Enabling the AIJ log server (ALS) process for a database improves performance for databases with medium to high update activity. However, a database with the ALS process enabled still experiences minimal stalls for AIJ write operations and commit operations. To further reduce the duration of these minimal stalls, you can create a file called an AIJ cache on an electronic disk to use as a temporary cache for AIJ write operations.

Electronic disks are solid-state disks, which have memory with battery backup; these disks function like a stable magnetic disk. They offer better performance than magnetic disks. Although it is possible to create an AIJ cache for an ALS process on a magnetic disk, you must create the AIJ cache on an electronic disk to get improved performance from this feature.

To enable and disable an AIJ cache on an electronic disk, use the SQL ALTER DATABASE statement, as shown in Example 3–11.

Example 3–11 Enabling and Disabling the AIJ Cache for an ALS Process

```
SQL> -- Enabling the AIJ cache for an ALS process
SQL> ALTER DATABASE FILENAME 'SQL_DISK1:[RICK]mf_personnel'
cont> JOURNAL IS ENABLED
cont> (CACHE FILENAME 'SQL_DISK2:[RICK]pers_cache');
```

(continued on next page)

Example 3–11 (Cont.) Enabling and Disabling the AIJ Cache for an ALS Process

```
SQL> --
SQL> -- Disabling the AIJ cache for an ALS process
SQL> ALTER DATABASE FILENAME mf_personnel
cont> JOURNAL IS ENABLED
cont> (NO CACHE FILENAME);
```

Enabling and disabling the AIJ cache on an electronic disk for an ALS process is an offline operation, which means it can be done only when there are no users attached to the database. Also, you cannot enable or disable the AIJ cache if the ALS process for the database is active (which can be determined by using the RMU Show Users command, as described in Section 3.5.1).

The SQL SHOW DATABASE statement and the Performance Monitor Journaling Information screen display the name of the cache file when an AIJ cache on an electronic disk for an AIJ process is enabled.

An AIJ cache on an electronic disk for an ALS process is partitioned into 64 blocks on each node in the cluster. The ALS processes on individual nodes in a cluster use different portions of the same AIJ cache.

An AIJ cache on an electronic disk for an ALS process improves the response time and throughput of database applications, while taking up only a small amount of space. It works in a cluster, does not need to be backed up, and is recovered automatically as part of the automatic database recovery (DBR) process if the ALS process fails.

3.5.3 Improving Performance by Disabling After-Image Journaling for WORM Storage Areas

Oracle Rdb allows users to store list (segmented string) data in storage areas on write-once, read-many (WORM) devices. By default, when databases have after-image journaling enabled and data is written to a WORM area (a storage area on a WORM device), Oracle Rdb also logs this information to the .aij file. Because information in a WORM area will never be overwritten, some users consider the logging of WORM changes to the .aij file to be unnecessary overhead. Such users would prefer to disable the logging of WORM changes to the .aij file, especially when loading data into WORM areas.

Disabling the writing of WORM changes to the .aij file also decreases the time spent rolling forward the database after a failure. The WORM area's journal records in the .aij file need to be applied even though the information in the WORM area is never overwritten. This increases the time spent recovering the database after a failure.

You can disable AIJ logging for write operations to WORM areas using the SQL ALTER DATABASE or CREATE DATABASE statements. The ALTER DATABASE statement shown in Example 3–12 disables AIJ logging of write operations for the WORM area 1986_EVENTS in the marketing database. This is an offline operation (it can be done only when there are no users attached to the database).

Example 3–12 Disabling AIJ Logging for WORM Area Write Operations

```
SQL> ALTER DATABASE FILENAME marketing
cont> ALTER STORAGE AREA 1986_EVENTS
cont> WRITE ONCE (JOURNAL IS DISABLED);
```

Note

> If you disable AIJ logging for write operations to WORM areas, the performance of applications using those WORM areas should improve. However, because data written to the WORM area is not being logged to the .aij file, there is no guaranteed way of being able to recover from WORM media failures. Before you disable AIJ logging for write operations to a WORM storage area to improve performance, you need to carefully consider the possibility that this action could result in lost data in the WORM area if the WORM media fails.

For example, suppose you have a WORM area for which AIJ logging has been disabled. Suppose that 120 pages were allocated in the WORM area, 100 pages were written to the area, and the last backup of the area was done when the area had 50 pages of information. If the WORM device fails, you will need to restore the WORM area from the last backup to new WORM media. The first 50 pages that were backed up earlier will be restored. Although the database may contain pointers to the WORM area between pages 50 and 120, the data for those pages is lost. In this database, an exception will be raised each time a request is made to fetch a page in the WORM area that was not restored.

Also, consider the case in which AIJ logging for write operations to a WORM area has been enabled or disabled more than once and the WORM area has not been properly backed up. If the WORM device fails, after the restore and rollforward operation completes, the WORM area will have several holes corresponding to regions that were not logged. An exception will be raised whenever a reference is made to one of these holes.

Disabling AIJ logging for a WORM area is a trade-off between the gain in performance from this action and the loss of data reliability (if a WORM device fails). If you disable AIJ logging for a WORM area, the following actions can help guard against the loss of data in WORM areas:

- Use other devices to shadow the WORM devices.
- Perform regular backups of the WORM storage areas.
- Keep multimedia objects written to WORM devices on other media until a backup is performed (for example, if you scan an image and store the data in a WORM area, keep the original image until the WORM area has been backed up).

3.6 Improving the Performance of the RMU Optimize After_Journal Command

You can improve the performance of the RMU Optimize After_Journal command by defining the RDM$BIND_OPTIMIZE_AIJ_RECLEN logical name or the RDB_BIND_OPTIMIZE_AIJ_RECLEN configuration parameter.

If you issue the RMU Optimize After_Journal command with the Log qualifier, a message similar to the following is displayed:

```
%RMU-I-OPTRECLEN, AIJ optimization record length was XXX characters in length
```

The XXX indicates the largest after-image journal optimization record whose size is smaller that the maximum configurable record size. The maximum configurable record size is 1536 characters by default. The optimized record length may be in the range of 512 to 4096 characters.

OpenVMS VAX OpenVMS Alpha

Consider the following OpenVMS example:

```
$ RMU/OPTIMIZE/AFTER/LOG backup1.aij opt.aij
%RMU-I-LOGOPNAIJ, opened journal file DISK$:[USER.WORK.AIJ]BACKUP1.AIJ;1
%RMU-I-LOGCREOPT, created optimized after-image journal file
DISK$:[USER.WORK.AIJ]OPT.AIJ;40
%RMU-I-OPTRECLEN, AIJ optimization record length was 474 characters in length
%RMU-S-AIJOPTSUC, AIJ optimization completed successfully
%RMU-I-LOGSUMMARY, total 8203 transactions committed
%RMU-I-LOGSUMMARY, total 0 transactions rolled back
```

♦

Use the value of the after-image journal optimization record length displayed in the message for the value of the RDM$BIND_OPTIMIZE_AIJ_RECLEN logical name or the RDB_BIND_OPTIMIZE_AIJ_RECLEN configuration parameter to optimize performance (474 in this example). You cannot predict the appropriate value before the optimization operation, but a given application is likely to generate comparable .aij record sizes. The value of the previous optimization operation is likely to be applicable to the next; a message will be displayed if the selected value is too small.

In general, Oracle Corporation recommends the following guidelines to improve the overall after-image journal optimization performance:

- Always use the RDMS$BIND_SORT_WORKFILES logical name or the RDB_BIND_SORT_WORKFILES configuration parameter to specify the number of work files you wish to use.
- Always use the SORTWORKn logical names to specify the file names of temporary work files on an unused, preferably fast, device.
- Never put two or more work files on the same device.
- Use fewer work files if the work file devices have lots of free space. Use more work files if available free space is limited.
- Do not use the Trace qualifier if you do not need to trace the output. The trace output greatly increases the number of buffered and direct I/Os and the overall elapsed time.
- Use the Log qualifier to obtain the OPTRECLEN message.
- Use the output of the OPTRECLEN message plus 10 percent as the value of the RDM$BIND_OPTIMIZE_AIJ_RECLEN logical name or the RDB_BIND_OPTIMIZE_AIJ_RECLEN configuration parameter; the minimum value is 512 and the maximum value is 4096.

- Do not optimize an .aij file containing many DDL records or .aij records greater than the RDM$BIND_OPTIMIZE_AIJ_RECLEN value; these types of records require an immediate sort and flush operation, which is extremely expensive and generates larger output files. The exact number of records depends largely on the application and the overall size of the input after-image journal.

DDL records and after-image journal records whose size is greater than the maximum do not prohibit the generation of the optimized .aij output file. They merely inhibit optimal performance of the optimization operation.

A message similar to the following is displayed if an .aij record is encountered that forces a sort and flush operation:

```
%RMU-I-OPTEXCMAX, TSN XXX record size YYY exceeds maximum ZZZ record size
```

If an .aij record contains a DDL operation that forces a sort and flush operation, a message similar to the following will be displayed:

```
%RMU-I-OPTDDLREC, TSN XXX contains DDL information that cannot be optimized
```

Also, if the Trace qualifier is specified, each internal sort operation will display a set of statistics information about the sort operation. For example:

```
%RMU-I-OPTSRTSTAT, Number of input records: 10048
%RMU-I-OPTSRTSTAT, Number of sorted records: 10048
%RMU-I-OPTSRTSTAT, Number of output records: 0
%RMU-I-OPTSRTSTAT, Number of merge passes: 1
%RMU-I-OPTSRTSTAT, Number of sort nodes: 6490
%RMU-I-OPTSRTSTAT, Workfile allocation blocks: 29675
```

To emphasize the importance of the RDM$BIND_OPTIMIZE_AIJ_RECLEN logical name and the RDB_BIND_OPTIMIZE_AIJ_RECLEN configuration parameter, consider the difference between the following identical tests:

Using RDM$BIND_OPTIMIZE_AIJ_RECLEN=4096:

```
 Direct I/O count :               7816
 Elapsed CPU time :       0 00:01:29.91
 Connect time :           0 00:04:17.31
```

Using RDM$BIND_OPTIMIZE_AIJ_RECLEN=512:

```
 Direct I/O count :                925
 Elapsed CPU time :       0 00:00:37.70
 Connect time :           0 00:01:25.59
```

If the value of the .aij optimization record length is 4096, then set the RDM$BIND_OPTIMIZE_AIJ_RECLEN logical name or the RDB_BIND_OPTIMIZE_AIJ_RECLEN configuration parameter to 4096. However, setting the value too large may decrease system performance.

3.7 Constraint Optimizations

Several types of constraint evaluation operations are optimized in Oracle Rdb, including:

- Existence
- Uniqueness
- Modification
- Database key (dbkey) retrieval and erasing

The following sections describe the nature of these optimizations. Also, refer to Section 8.1.3.1 for additional information on constraints and performance.

3.7.1 Existence Constraint

The existence constraint is not evaluated when you store rows in or delete rows from tables named in the constraint. For example:

```
SQL> ALTER TABLE SALARY_HISTORY
cont> ADD CONSTRAINT CONSTRAINT SH_EMP_ID_EXISTS
cont> CHECK (EMPLOYEE_ID = ANY (SELECT EMPLOYEE_ID
cont> FROM EMPLOYEES))
cont> DEFERRABLE;
```

Oracle Rdb need not evaluate this constraint when you store rows in the EMPLOYEES table or delete rows from the SALARY_HISTORY table.

3.7.2 Uniqueness Constraint

The uniqueness constraint is not evaluated when a row in a table named in the constraint is deleted from the database. For example:

```
SQL> ALTER TABLE EMPLOYEES
cont> ALTER EMPLOYEE_ID COL1 CONSTRAINT EMP_UNIQUE
cont> UNIQUE DEFERRABLE;
```

The EMP_UNIQUE constraint does not have to be evaluated when you delete rows from the EMPLOYEES table.

3.7.3 Modification Operation

If the columns in the table you modify are not named in the constraint definition, Oracle Rdb need not evaluate the constraint. For example:

```
SQL> ALTER TABLE EMPLOYEES
cont> ALTER EMPLOYEE_ID CONSTRAINT EMP_ID_RANGE
cont> CHECK (EMPLOYEE_ID > '00000')
cont> NOT DEFERRABLE;
```

If you modify columns other than EMPLOYEE_ID in the EMPLOYEES table, Oracle Rdb does not have to evaluate this constraint.

If any column named in the constraint is a COMPUTED BY field, the constraint always is evaluated whenever you modify any column in the table.

If the modification is an equivalence statement, the constraint does not have to be evaluated. For example, Oracle Rdb does not evaluate the constraint when the following query executes:

```
SQL> UPDATE EMPLOYEES
cont> SET EMPLOYEE_ID = EMPLOYEE_ID;
```

3.7.4 Database Key Retrieval and Erasing

Consider the following constraint definition:

```
SQL> ALTER TABLE EMPLOYEES
cont> ADD CONSTRAINT CONSTRAINT EMP_EXIST_SH
cont> CHECK (EMPLOYEE_ID = ANY (SELECT EMPLOYEE_ID
cont> FROM SALARY_HISTORY))
cont> DEFERRABLE;
```

Oracle Rdb uses database keys to locate rows in the database for retrieval. In this example, any query executed for the EMPLOYEES table processes rows in that table one row at a time. Because Oracle Rdb already has dbkeys for each row it processes, the constraint evaluation mechanism can use these database keys to retrieve single rows for constraint evaluation.

3.8 Locking

One of the primary features of a database management system is its ability to ensure data consistency. When many users access the database at the same time, a column's value may be modified several times. After all modifications to that column have been completed, you should expect the final value assigned to that column to be the most current and correct value. If no mechanism were in place to protect the current value of a retrieved column from corruption by another user, you would have little confidence in any values in the database. Oracle Rdb uses a lock manager[1] to synchronize access of shared resources.

Lock conflict between users is a cause of performance problems. Lock conflicts occur when more than one transaction attempts to lock the same resource at the same time. You can use the Performance Monitor to display statistics on:

- Locks (particular lock type)

[1] On OpenVMS, Oracle Rdb uses the OpenVMS lock manager. On Digital UNIX, Oracle Rdb uses the Oracle Rdb lock manager.

- Summary locking statistics (locks requested, locks promoted, locks demoted, locks released, blocking asynchronous system traps (ASTs), stall time, and invalid lock block)
- Lock statistics for one statistics field (for example, locks requested)
- Lock statistics (by file)

You can use the RMU Show Locks command to display information about process locking activity.

OpenVMS VAX OpenVMS Alpha

The OpenVMS MONITOR LOCK command can also help you determine the cause of database lock problems. See the OpenVMS documentation set for more information. ♦

Section 3.8.1 describes the tools you can use to gather locking information. Subsequent sections in this chapter describe lock considerations that can affect database performance. Section 8.4 provides a step-by-step check for diagnosing lock problems.

3.8.1 Gathering Lock Information

Sections 3.8.1.1 through 3.8.1.6 describe RMU Show commands, Performance Monitor screens, and the System Dump Analyzer (SDA) utility[1], all of which can be used to gather lock information. Stall messages, described in Section 3.2.1.3 and Section 3.2.1.5, also provide locking information. Refer to Section 8.4 for more advice on analyzing locks.

3.8.1.1 RMU Show Locks Command

The RMU Show Locks command displays information about process locking activity and lock contention for all active databases on a specific node.

In a clustered system, the RMU Show Locks command normally displays detailed information for your current node only, although you can retrieve some general, clusterwide information.

OpenVMS VAX OpenVMS Alpha

To use the RMU Show Locks command on an OpenVMS system, you must have the OpenVMS process privilege WORLD or a higher privilege. ♦

[1] On OpenVMS systems only

Digital UNIX

To use the RMU Show Locks command on a Digital UNIX system, you must be the dbsmgr user or the superuser. ♦

Table 3–6 shows the RMU Show Locks command qualifiers and provides a brief description of each. Further information is supplied in the text accompanying the examples at the end of this section. For a detailed description of the RMU Show Locks command, refer to the *Oracle RMU Reference Manual*.

Table 3–6 RMU Show Locks Command Qualifiers

Qualifier	Description
Lock=(lock-id [, lock-id ...])	Displays locking information for a specified lock ID or list of lock IDs. If you specify more than one lock, the IDs must be enclosed in parentheses and separated by commas. A lock ID is an 8-digit, hexadecimal number that must be local to the node on which the RMU Show Locks command is issued. You can use the Performance Monitor Stall Messages screen to show the lock ID that is causing a process to wait.

(continued on next page)

Table 3–6 (Cont.) RMU Show Locks Command Qualifiers

Qualifier	Description
Mode = mode-list	Indicates the lock mode (Blocking or Waiting) that you want to display. • The Blocking mode option displays processes whose locks are blocking the lock requests of other processes. The first line of output identifies a process that is waiting for a lock request to be granted. Subsequent lines of output identify those processes that are preventing the lock request from being granted. When multiple processes are waiting for the same lock resource, multiple sets of process-specific information, one for each waiting process, are displayed. • The Waiting mode option displays the set of processes whose lock requests are waiting due to conflicting locks that have been granted to other processes. The first line of output identifies the process that has been granted a resource lock. Subsequent lines of output identify those processes that are waiting for a lock on the same resource. When multiple processes are blocking the same lock resource, multiple sets of process-specific information, one for each blocking process, are displayed. If you specify both Blocking and Waiting, you must separate the options with a comma and enclose them in parentheses.

(continued on next page)

Table 3–6 (Cont.) RMU Show Locks Command Qualifiers

Qualifier	Description
Options = options-list	Indicates the type of information and the level of detail the output should include. The two Options values are: All and Full. • Use the All option to display lock information for all other processes that also have an interest in the lock held by the specified process. The All option must be used with the Process qualifier and without the Mode qualifier. • Use the Full option to display lock information about special database processes. A number of special database processes, such as monitors, perform work on behalf of a database. These database processes frequently request locks that, by design, conflict with other processes' locks. If you specify more than one Options value, you must separate the values with a comma and enclose the options-list in parentheses.
Output = file-name	Specifies the name of the file where output is sent. If you do not specify a file type, the default output file type is .lis. If you do not specify a file name, output is sent to SYS$OUTPUT on OpenVMS and the standard output (stdout) device on Digital UNIX.
Process = (process-id [, process-id ...])	Displays locking information for the specified process ID or list of process IDs. If you specify more than one process, the IDs must be enclosed in parentheses and separated by commas.

If you do not specify any qualifiers, RMU Show Locks displays all the locks on your node. This display can be large.

You can halt the output from the RMU Show Locks command by pressing Ctrl/S and then restart the halted output by pressing Ctrl/Q. If you decide that you want to completely terminate the RMU Show Locks command, you can do this on OpenVMS by pressing either Ctrl/C or Ctrl/Y and on Digital UNIX by pressing Ctrl/C.

Table 3–7 indicates the actions that result from several different combinations of the command qualifiers; the table does not show all the possible combinations. You can specify the Options qualifier using the Full value with every combination shown in the table without affecting the action. Therefore, Options=Full is not included in the table.

Table 3–7 RMU Show Locks Command Qualifier Combinations

If You Specify RMU Show Locks with . . .	You Will See Lock Information for . . .
Process=process-id and Mode=Blocking	All the processes blocking the specified process or list of processes
Process=process-id and Mode=Waiting	All the processes waiting for the specified process or list of processes
Process=process-id and Options=All	All the processes that have an interest in the lock held by the specified process or list of processes
Lock=lock-id and Mode=Blocking	All the processes blocking the specified lock or list of locks
Lock=lock-id and Mode=Waiting	All the processes waiting for the specified lock or list of locks
Mode=(Blocking,Waiting) and Process=(id-1,id-2)	All the processes blocking process id-1, and all the processes waiting for process id-2

When you specify a list of process or lock identifiers, make sure the processes or locks are local to the node on which the RMU Show Locks command is issued.

The rest of this section provides sample commands and output from several RMU Show Locks qualifier combinations. Each example shows output that results from contention between two transactions. The first transaction is associated with process ID 44A047C9:

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ WRITE RESERVING EMPLOYEES FOR EXCLUSIVE WRITE;
```

The second transaction is associated with process ID 44A045D1:

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SELECT * FROM EMPLOYEES WHERE LAST_NAME='Toliver';
```

The second transaction must wait for the first transaction to commit or roll back before it can execute. The first transaction has exclusive access to the EMPLOYEES table, which the second transaction needs to read.

The heading for each report that you generate indicates what qualifiers were used in the command line. Refer to the *Oracle RMU Reference Manual* for an explanation of the report format and content. Refer to Table 3–8 for a description of the lock types that can appear in the Requested and Granted queues. Note that the SR and SW locks shown in Table 3–8 are equivalent to the CR and CW locks, respectively, that can appear in a RMU Show Locks report. Also, the NL (NULL) lock appears in the report but not in the table.

The NL lock is used to indicate an interest in the resource, or as a placeholder for future lock conversions.

Example 3–13 shows a portion of the output generated by the RMU Show Locks command with the Process=44A047C9 qualifier. The actual report is several pages long because it shows all the locks held by process ID 44A047C9. The report text shows the resource on which the lock is held, ID information, and lock status (Requested and Granted).

Example 3–13 Displaying Locks for a Process

```
================================================================================
SHOW LOCKS/PROCESS Information
================================================================================
   .
   .
   .
--------------------------------------------------------------------------------
Resource: page 352
          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Owner:    44A047C9  USER1..........     7CC80BC8  00020025  PR        PR

--------------------------------------------------------------------------------
Resource: cluster membership
          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Owner:    44A047C9  USER1..........     16180C1A  00020025  PR        PR

   .
   .
   .
--------------------------------------------------------------------------------
Resource: logical area 39
          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Owner:    44A047C9  USER1..........     45983EC0  00020025  EX        EX

   .
   .
   .
```

(continued on next page)

Example 3–13 (Cont.) Displaying Locks for a Process

```
--------------------------------------------------------------------------------
Resource: logical area 33

          ProcessID Process Name       Lock ID   System ID Requested Granted
          --------- ---------------    --------- --------- --------- -------
Owner:    44A047C9  USER1..........    0480973C  00020025  CR        NL

--------------------------------------------------------------------------------
Resource: logical area 53

          ProcessID Process Name       Lock ID   System ID Requested Granted
          --------- ---------------    --------- --------- --------- -------
Owner:    44A047C9  USER1..........    56009774  00020025  EX        EX

   .
   .
   .
```

Example 3–14 shows the output generated by the RMU Show Locks command with the Process=44A047C9 and the Mode=Waiting qualifiers. The report shows that process ID 44A045D1 is waiting for the exclusive lock held on logical area 39 by the specified process (44A047C9) to be released.

This command identifies the waiting process. If you specify the ID of a process that is itself a waiting process, Oracle RMU returns the following message: `no locks on this node with the specified qualifiers.`

Example 3–14 Identifying Processes Waiting for Locks

```
================================================================================
SHOW LOCKS/PROCESS/WAITING Information
================================================================================

--------------------------------------------------------------------------------
Resource: logical area 39

          ProcessID Process Name       Lock ID   System ID Requested Granted
          --------- ---------------    --------- --------- --------- -------
Blocker:  44A047C9  USER1..........    45983EC0  00020025  EX        EX
Waiting:  44A045D1  _RTA11:........    3B5467DA  00020025  CR        NL
```

Example 3–15 shows the output generated by the RMU Show Locks command with the Process=44A045D1 and Mode=Blocking qualifiers. The report shows that process ID 44A047C9 has an exclusive lock on logical area 39, and is blocking the specified process (44A045D1).

This command identifies the blocking process. If you specify the ID of a process that is itself the blocking process, Oracle RMU returns the following message: `no locks on this node with the specified qualifiers.`

Example 3–15 Identifying Blocking Processes

```
================================================================================
SHOW LOCKS/BLOCKING Information
================================================================================

--------------------------------------------------------------------------------
Resource: logical area 39

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Waiting:  44A045D1  _RTA11:........     3B5467DA  00020025  CR        NL
Blocker:  44A047C9  USER1..........     45983EC0  00020025  EX        EX
```

Example 3–16 shows the output generated by the RMU Show Locks command with the Lock=45983EC0 and Mode=Waiting qualifiers. The report is identical to the display shown in Example 3–14 because process ID 44A047C9 has taken out only one lock. If process ID 44A047C9 held multiple locks, Example 3–14 would display all of them, but this example would only display lock information for lock ID 45983EC0.

Example 3–16 Displaying Lock IDs for Waiting Processes

```
================================================================================
SHOW LOCKS/LOCK/WAITING Information
================================================================================

--------------------------------------------------------------------------------
Resource: logical area 39

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Blocker:  44A047C9  USER1..........     45983EC0  00020025  EX        EX
Waiting:  44A045D1  _RTA11:........     3B5467DA  00020025  CR        NL
```

Example 3–17 shows a portion of the output generated by the RMU Show Locks command with the Process=44A047C9 and Options=All qualifiers. The full report is several pages long because it lists all the resources that have locks held by process ID 44A047C9 and all the locks on the same resource held by other processes. Compare this report with the one shown in Example 3–13.

Example 3–17 Identifying All Resources Held by a Process Lock

```
================================================================================
SHOW LOCKS/PROCESS Information
================================================================================
   .
   .
   .
--------------------------------------------------------------------------------
Resource: page 352

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Owner:    44A047C9  USER1..........     7CC80BC8  00020025  PR        PR
Owner:    44A045D1  _RTA11:........     134C0979  00020025  PR        CR
--------------------------------------------------------------------------------
Resource: cluster membership

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Owner:    44A047C9  USER1..........     16180C1A  00020025  PR        PR
Owner:    44A045D1  _RTA11:........     333C95ED  00020025  PR        PR

   .
   .
   .
--------------------------------------------------------------------------------
Resource: logical area 39

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Owner:    44A047C9  USER1..........     45983EC0  00020025  EX        EX
Waiting:  44A045D1  _RTA11:........     3B5467DA  00020025  CR        NL

   .
   .
   .
--------------------------------------------------------------------------------
Resource: logical area 33

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Owner:    44A047C9  USER1..........     0480973C  00020025  CR        NL
Owner:    44A045D1  _RTA11:........     31900BAE  00020025  CR        CR
--------------------------------------------------------------------------------
Resource: logical area 53

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Owner:    44A047C9  USER1..........     56009774  00020025  EX        EX

   .
   .
   .
```

Example 3–18 shows the output generated by the RMU Show Locks command with the Mode=(Waiting,Blocking) and the Process=(44A045D1,44A047C9) qualifiers. This command combines the output shown in Example 3–15 and Example 3–16 into a single report.

Example 3–18 Identifying Waiting and Blocked Processes

```
================================================================================
SHOW LOCKS/PROCESS/BLOCKING Information
================================================================================

--------------------------------------------------------------------------------
Resource: logical area 39

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Waiting:  44A045D1  _RTA11:........     3B5467DA  00020025  CR        NL
Blocker:  44A047C9  USER1..........     45983EC0  00020025  EX        EX

================================================================================
SHOW LOCKS/PROCESS/WAITING Information
================================================================================

--------------------------------------------------------------------------------
Resource: logical area 39

          ProcessID Process Name        Lock ID   System ID Requested Granted
          --------- ---------------     --------- --------- --------- -------
Blocker:  44A047C9  USER1..........     45983EC0  00020025  EX        EX
Waiting:  44A045D1  _RTA11:........     3B5467DA  00020025  CR        NL
```

Interpreting the Meaning of the Granted and Requested Modes in RMU Show Locks Output Displays

When displaying locks using the RMU Show Locks command, the “Requested” and “Granted” modes of the given lock are displayed. The definitions of these modes are:

- Requested

  This is the mode for which the process has requested the lock. Valid modes are NL, PR, PW, CR, CW, and EX. This mode is not guaranteed to be granted; some locks are intentionally held in conflicting modes forever (for example, the termination lock).

- Granted

  This is the mode that the process was last granted for the lock. Valid modes are NL, PR, PW, CR, CW, and EX. If the lock has never been previously granted, the lock mode is displayed as NL mode.

If the Requested and Granted lock modes are different in the RMU Show Locks output, then the lock requested is currently blocked on either the Waiting or Conversion queue. If the modes are the same, then the lock has been granted.

The lock manager does not always update the Requested lock mode. This means that potentially conflicting information can be displayed by the RMU Show Locks command.

The Requested lock mode is only updated under the following situations:

1. The lock request is for a remote resource.
2. The lock request is a Nowait request.
3. The lock request could not be granted due to a lock conflict (that is, it was canceled by the application or aborted due to lock timeout/deadlock).
4. The lock request is the first for the resource.

Example 3–19 shows a portion of the output from the RMU Show Locks command.

Example 3–19 RMU Show Locks

```
$ RMU/SHOW LOCKS
================================================================================
          SHOW LOCKS Information
================================================================================
   .
   .
   .
--------------------------------------------------------------------------------
Resource Name: page 533
Granted Lock Count: 1,  Parent Lock ID: 01000B6C,   Lock Access Mode: Executive,
Resource Type: Global,  Lock Value Block: 03000000 00000000 00000000 00000002

          -Master Node Info-  --Lock Mode Information--     -Remote Node Info-
ProcessID Lock ID   SystemID  Requested Granted   Queue     Lock ID   SystemID
2040021E  0400136A  00010002  EX        CR        GRANT     0400136A  00010002
--------------------------------------------------------------------------------
   .
   .
   .
```

It is ordinarily difficult to explain how the combination of lock modes shown in Example 3–19 could occur. Note that the CR (concurrent read) mode is on the Grant queue (not the Conversion queue).

Knowledge of the operating environment is necessary to know that there was only one node on this system. Two lock requests occurred to generate this output, in the opposite order of what appears to have occurred.

The first lock request was for EX (exclusive) mode, which was immediately granted. Thus, the Requested and Granted modes were updated according to situation 4. Then, the lock was demoted from EX to CR mode, which was also immediately granted. However, the Requested field was not updated because none of the four preceding rules was true, so the Requested mode was never updated to reflect the CR lock request.

3.8.1.2 Displaying Integrated Lock Information

The Performance Monitor provides integrated lock information on the Stall Messages, Active User Stall Messages, and DBR Activity screens.

Type L to display a menu of any lock IDs currently displayed on the active screen.

For example, consider the following Performance Monitor Active User Stall Messages screen:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:48:03
Rate: 1.00 Second          Active User Stall Messages       Elapsed: 03:06:07.20
Page: 1 of 1     KODD$:[R_ANDERSON.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID Since......   Stall.reason............................. Lock.ID.
2B600554:1 15:38:27.04 - waiting for record 55:321:2 (EX)          1500624B
2B600556:1 15:40:40.66 - waiting for record 55:321:2 (EX)          1200A2E4
2B600555:1               writing pages back to database

--------------------------------------------------------------------------------
Config Exit Help LockID Menu >next_page <prev_page Set_rate Write Zoom !
```

Typing L displays the menu of available lock IDs. For example:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:48:11
Rate: 1.00 Second          Active User Stall Messages       Elapsed: 03:06:07.20
Page: 1 of 1     KODD$:[R_ANDERSON.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID Since......   Stall.reason............................. Lock.ID.
2B600554:1 15:38:27.04 - waiting for record 55:321:2 (EX)          A. 1500624B
2B600556:1 15:40:40.66 - waiting for record 55:321:2 (EX)          B. 1200A2E4
2B600555:1               writing pages back to database

--------------------------------------------------------------------------------
Config Exit Help LockID Menu >next_page <prev_page Set_rate Write Zoom !
```

Note the displayed lock IDs have been prefixed with the standard menu-selection letter choice. The desired lock ID can be selected by either typing the desired letter, or moving the highlighted cursor using the up arrow or down arrow keys and then pressing the Return key.

If there are no lock IDs on the display, typing L has no effect.

When the desired lock ID is selected, the Lock Information display appears. For example:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:48:15
Rate: 1.00 Second          Active User Stall Messages      Elapsed: 03:06:07.20
Page: 1 of 1     KODD$:[R_ANDERSON.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID Since......   Stall.reason............................. Lock.ID.
2B600554:1 15:38:27.04 - waiting for record 55:321:2 (EX)        >1500624B<
2B600556:1 15:40:40.66 - waiting for record 55:321:2 (EX)         1200A2E4
2B600555:1               writing pages back to database

+-- Lock Information: 1500624B ---------------------------+
|                                                         |
|              Resource: record 321:2                     |
| State... ProcessID Process.Name... Lock.ID.  Rq Gr Queue |
|                                                         |
| Blocker: 2B600555  RICK3.......... 0F005AB6  EX EX Grant |
| Waiting: 2B600556  RICK4.......... 1200A2E4  EX NL Cnvrt |
| Waiting: 2B600554  RICK2.......... 1500624B  EX NL Cnvrt |
|                                                         |
+---------------------------------------------------------+

--------------------------------------------------------------------------------
Config Exit Help LockID Menu >next_page <prev_page Set_rate Write Zoom !
```

The Lock Information display is not dynamic. The lock information is captured at the time the lock ID is selected, and the information does not dynamically change. The Lock Information display is a snapshot in time of the selected lock ID.

The Lock Information display can only be displayed. It cannot be saved using the Write option.

The Lock Information display provides the following information:

Information	Status
State	This field indicates whether the respective lock is an owner, a blocker, or is waiting for the resource.
ProcessID	This field indicates the ID of the process that owns the respective lock.
Process.Name	This field indicates the name of the process that owns the respective lock.
Lock.ID	This field indicates the ID of the lock on the indicated resource.
Rq	This field indicates the mode in which the respective lock was requested. Modes are NL, CR, CW, PR, PW, and EX.

Information	Status
Gr	This field indicates the mode in which the respective lock was granted. Modes are NL, CR, CW, PR, PW, and EX. The granted mode may differ from the requested mode if the lock is on the conversion queue or if the lock was demoted from a stronger to a weaker mode.
Queue	This field indicates the queue on which the respective lock resides. There are three queues: • Grant —the Grant queue • Cnvrt—the Conversion queue • Wait—the Waiting queue

Note that while the order of displayed locks is arbitrary, you can identify the order in which waiting locks will be granted. For instance, in the previous example, two processes are waiting for the same lock. Displaying the information for lock 1500624B simply indicates that both processes are waiting for the same resource. Displaying the information for lock 1200A2E4 shows the following information:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:49:01
Rate: 1.00 Second         Active User Stall Messages      Elapsed: 03:06:07.20
Page: 1 of 1    KODD$:[R_ANDERSON.WORK.AIJ]MF_PERSONNEL.RDB;1     Mode: Online
--------------------------------------------------------------------------------
Process.ID Since......   Stall.reason............................ Lock.ID.
2B600554:1 15:38:27.04 - waiting for record 55:321:2 (EX)         1500624B
2B600556:1 15:40:40.66 - waiting for record 55:321:2 (EX)        >1200A2E4<
2B600555:1               writing pages back to database

+-- Lock Information: 1500624B ---------------------------+
|                                                         |
|             Resource: record 321:2                      |
| State... ProcessID Process.Name... Lock.ID.  Rq Gr Queue |
|                                                         |
| Blocker: 2B600555  RICK3.......... 0F005AB6  EX EX Grant |
| Blocker: 2B600554  RICK2.......... 1500624B  EX NL Cnvrt |
| Waiting: 2B600556  RICK4.......... 1200A2E4  EX NL Cnvrt |
|                                                         |
+---------------------------------------------------------+

--------------------------------------------------------------------------------
Config Exit Help LockID Menu >next_page <prev_page Set_rate Write Zoom !
```

This display indicates that, although lock 1200A2E4 is on the conversion queue, it is blocking the other lock and will be granted before the other lock.

The Lock Information display can contain more information than can be displayed on the terminal screen. Therefore, you can scroll through the Lock Information display either one line at a time or one screen at a time.

You can move the lock information up or down one line at a time using the up arrow or down arrow keys. You can also move the lock information up or down one page at a time using the Previous Screen or left arrow key, or the Next Screen or right arrow key. These keys are activated when the indicators "<–MORE" and/or "MORE–>" dynamically appear on the lock information display itself. The horizontal menu at the bottom of the screen also indicates whether these keys are activated. If the navigational keys are not activated, nothing happens if you attempt to use them.

Note that the Lock Information display size is based on the number of lines configured for the terminal. For instance, a terminal with 48 lines has a larger Lock Information display than a terminal with 24 lines.

Because the database is continuing to operate while you display the lock information, the selected lock can be released before its information can be displayed. If this happens, the message "Lock has been released since it was displayed" is displayed on the terminal screen. Also, the lock ID menu is canceled when you return from the Lock Information screen.

If you do not have sufficient privilege to retrieve the lock information, the message "Insufficient privilege to display lock information" is displayed on the terminal screen. Also, the lock ID menu is canceled when you return from the Lock Information screen.

Because of the dynamic nature of the lock information, the lock information is not written to the output file and cannot be replayed from an input file.

While the Lock Information display is displayed, terminal broadcast messages are suspended. The messages resume after you leave the Lock Information display.

Note that on the Active User Stall Messages screen, stalled locks that are no longer stalled are still displayed. Even if the displayed lock is no longer stalled, you can still attempt to display information on that stalled lock. However, the lock that was stalled is probably released.

The Lock Information display does not attempt to provide detailed information about the process holding a respective lock. The process information can be obtained using the Process Accounting screen if the process is on the current node.

Because of the current nonsplitting menu limitation of 36 options, using the menu-selection letters A to Z and 0 to 9, only the first 36 lock IDs on a given display are selectable. This restriction may be removed in the future. To work around this restriction, set the terminal screen length to less than 42 lines or use the other Stall Messages screen, where the order of the displayed lock IDs may be different from the current screen.

3.8.1.3 Gathering Lock Statistics with the Performance Monitor

The Performance Monitor provides the following screens to help gather information about locking:

- The Summary Locking Statistics screen shows a summary of the lock activity done by Oracle Rdb.
- The Locking for One Lock Type screen shows lock statistics for a particular lock type. You should use this screen when you want all the available information on a specific lock type. The only difference between this lock screen and the Lock Statistics for One Statistics Field screen is that these statistics pertain only to the type of lock that you specify and are, therefore, more useful for detailed analysis of lock activity.
- The Locking for One Statistics Field screen shows lock statistics for all the lock types of a particular statistical field. You should use this screen when you want to see specific information about all the different lock types.
- The Lock Deadlock History screen can be used to identify the object that causes a deadlock event.

  For more information, see Section 3.8.1.4.
- The Lock Timeout History screen can be used to identify the object that causes a timeout event.

  For more information, see Section 3.8.1.5.
- The Lock Statistics by file screen displays information about page locks that are specific to storage areas and snapshot files. This information is vital in determining which storage areas have the most locking activity, and analyzing the validity of storage area partitioning.

3.8.1.4 Performance Monitor Lock Deadlock History Screen

The Lock Deadlock History screen can be used to identify the object that causes a deadlock event. You access the Lock Deadlock History screen from the Process Information submenu.

The Stall Messages screen provides stall information, but when a lock is deadlocked, the stall is terminated and the information is no longer available on the Stall Messages screen.

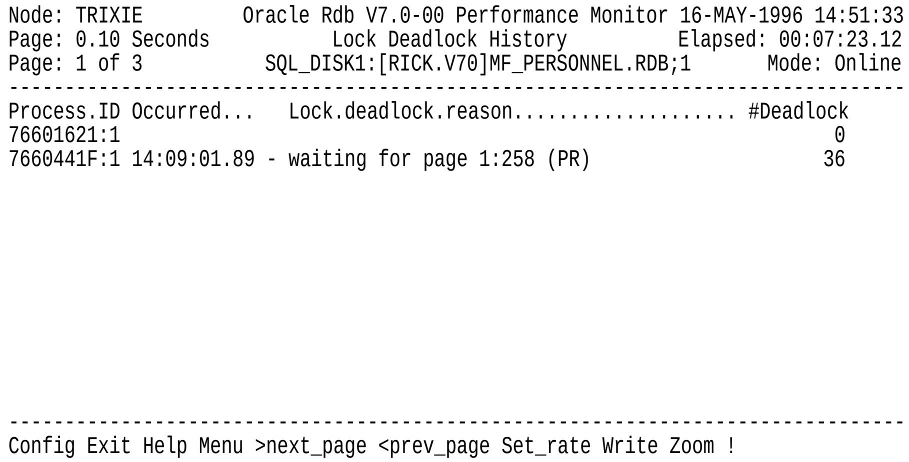
For each active process on the current node, the Lock Deadlock History screen shows the process ID, the time that the process was most recently involved in a deadlock, the reason for the most recent deadlock, and the number of deadlocks the process has encountered since attaching to the database.

The following Lock Deadlock History screen shows a record deadlock. The lock deadlock reason indicates that the process was waiting for a record and provides the dbkey of that record.

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:48:12
Page: 0.10 Seconds           Lock Deadlock History         Elapsed: 00:07:21.10
Page: 1 of 3           SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID Occurred...   Lock.timeout.reason...................... #Timeout
65505487:1 08:55:21.16   Waiting for record 1:2:1 (CR)                    1







--------------------------------------------------------------------------------
Config Exit Help Menu >next_page <prev_page Set_rate Write Zoom !
```

By using the Lock Deadlock History screen to examine the most recent deadlocks, a DBA can more easily identify potential database hot spots and application bottlenecks. Looking at a screenful of deadlock information can help the DBA differentiate between frequent and infrequent problems and correlate common causes of the problems.

The following Lock Deadlock History screen shows a page deadlock. Although a page deadlock is not necessarily a bad occurrence, it could indicate a potential performance problem. Note that process 7660441F has had 36 deadlocks, although not necessarily on the indicated page (the lock deadlock reason gives the reason for only the most recent deadlock). If 36 deadlocks is a high number for the time the process has been attached to the database, the DBA should examine the process in detail. A high number of deadlocks for a process can indicate a major design problem.



```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:51:33
Page: 0.10 Seconds          Lock Deadlock History         Elapsed: 00:07:23.12
Page: 1 of 3          SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID Occurred...   Lock.deadlock.reason................... #Deadlock
76601621:1                                                              0
7660441F:1 14:09:01.89 - waiting for page 1:258 (PR)                   36








--------------------------------------------------------------------------------
Config Exit Help Menu >next_page <prev_page Set_rate Write Zoom !
```

In general, record deadlocks are undesirable and the cause of such deadlocks should be discovered and fixed. Page deadlocks are not always bad, but they need to be analyzed.

3.8.1.5 Performance Monitor Lock Timeout History Screen

The Lock Timeout History screen can be used to identify the object that causes a timeout event. You access the Lock Timeout History screen from the Process Information submenu.

The Stall Messages screen provides stall information, but when a process times out while waiting for a lock, the stall is terminated and the information is no longer available on the Stall Messages screen.

For each active process on the current node, the Lock Timeout History screen shows the process ID, the time that the process most recently timed out while waiting for a lock, the reason for the most recent timeout experienced by the process, and the number of timeouts experienced by the process since attaching to the database.

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 16-MAY-1996 14:54:13
Rate: 3.00 seconds           Lock Timeout History         Elapsed: 00:00:31.75
Page: 1 of 3        SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
Process.ID Occurred...   Lock.timeout.reason...................... #Timeout
65505487:1 08:55:21.16   Waiting for record 1:2:1 (CR)                    1







--------------------------------------------------------------------------------
Config Exit Help Menu >next_page <prev_page Set_rate Write Zoom !
```

By examining the first Lock Deadlock History screen example in Section 3.8.1.4 and this Lock Timeout History screen, the DBA could determine that the record with a dbkey of 1:2:1 is a database hot spot and the source of contention.

3.8.1.6 System Dump Analyzer

OpenVMS VAX OpenVMS Alpha

To find out more about locks, you can use the System Dump Analyzer (SDA) utility. For more information, see the OpenVMS documentation set.

To use SDA, take the following steps:

1. You must have the OpenVMS CMKRNL privilege to use SDA. The CMKRNL privilege is powerful; it should not be given to users who do not need it.
2. Use the Stall Messages or Active User Stall Messages screens to isolate the process ID (PID) of the stalled process.
3. Invoke SDA, as follows:

   ```
   $ ANALYZE/SYSTEM
   ```

4. Get the index value of the PID of the stalled process:

   ```
   SDA> SHOW SUMMARY
   ```

5. Select the stalled process by PID:

   ```
   SDA> SET PROCESS/INDEX=index_value
   ```

6. Verify that you have the right process:

   ```
   SDA> SHOW PROCESS
   ```

7. Create an output file that you will be able to edit later:

   ```
   SDA> SET OUTPUT file_name.ext
   ```

8. Display the process information:

   ```
   SDA> SHOW PROCESS/INDEX=index_value/LOCKS
   ```

9. Close the file containing the process information by redirecting output of subsequent commands to SYS$OUTPUT:

   ```
   SDA> SET OUTPUT SYS$OUTPUT
   ```

10. Start a subprocess so that you can edit the file that you created with the lock information:

    ```
    SDA> SPAWN
    ```

11. Search, starting from the bottom of the file, for the text string “Waiting”:

    ```
    $ EDIT file_name.ext
    ```

12. Write down the lock ID for this particular lock and log out of the subprocess.

    ```
    $ LO   (get back to SDA)
    ```

13. Show information about the particular lock:

    ```
    SDA> SHOW RESOURCE/LOCK=lock_id
    ```

    This shows what locks are granted, what locks are waiting, and what resource is locked. Write down all LOCK QUEUE information.

14. Determine the PID value:

    ```
    SDA> SHOW LOCK lock_id
    ```

15. Locate a valid process ID value:

    ```
    SDA> SHOW PROCESS/INDEX=pid_from_previous_step
    ```

16. Exit:

    ```
    SDA> EXIT
    ```

17. Show which process has caused or is partially responsible for the stall:

    ```
    $ SHOW PROCESS/ID=Process_ID_value
    ```

This process may have to be repeated for any stalled processes.

If the problem is caused by a batch job, the logic in that software may have to be changed. In most cases, transactions should be started as NOWAIT, and have the software trap errors and schedule retries. Oracle Rdb defaults to transactions started as WAIT, which more than likely relate to a stall condition.

Use of sorted indexes and their associated index node sizes may also relate to the problem. Hashed indexes may be an effective alternative. Otherwise, consider reducing the index node size.

It is also possible that somebody suspended the process holding the locks, possibly by entering Ctrl/Y without a STOP or EXIT command, or by suspending a batch job. ♦

3.8.2 Lock Considerations

When one user's query fetches a row from the database in a read/write transaction, a series of protected read (PR) or protected write (PW) locks prevents other users from altering that row. Oracle Rdb places these locks not only on that row, but on database tables, pages, and index nodes as well. Any subsequent attempt to retrieve the row must not conflict with the current user's operation on that row. However, if a user's query fetches a row from the database in a read-only transaction, the query uses a shared read (SR) lock on the logical area so other users know a transaction is active in the logical area. No locking is done at the row level, but the read-only transaction uses page locking to ensure that the instance of the database page is valid in each user's cache. Refer to the *Oracle Rdb7 Guide to Database Design and Definition* for further information on Oracle Rdb locking activity.

Oracle Rdb controls access to database resources through transactions. When you begin a transaction, you specify the kind of database activity that will take place; you can restrict access to those tables you intend to use, or permit other users to share the same data. Once a transaction is started, all operations performed within the scope of a transaction are guaranteed to be consistent. That is, the rows a user retrieves or updates are stable for the duration of the transaction. When the transaction is terminated, updated rows become available to other database users. However, while the transaction is in progress, no intervening user can modify any of those rows.

By specifying a transaction mode that permits a set of database operations to complete successfully, the user can influence the type of concurrent database activity against which his or her transaction competes. If your application requires concurrent multiuser read access, enabling snapshot files and using read-only transactions prevents conflict with other transaction types except batch-update transactions and transactions that use the lock specification clause (RESERVING . . . FOR EXCLUSIVE . . .). For most other cases, read/write transactions are used where update access is necessary. Section 3.8.3 describes the database access modes you can include in the SQL SET TRANSACTION statement. Refer to the *Oracle Rdb7 SQL Reference Manual* for details on the SQL SET TRANSACTION statement and detailed information on lock compatibility among the different types of transactions.

3.8.3 Reserving Options

The best transaction design keeps the amount of locking to a minimum. By using a reserving option in your SQL SET TRANSACTION statement, you can attain the level of database concurrency or security your tasks require. A reserving option is the combination of a share mode and a lock type (read or write). The share mode options are:

- Shared
- Protected
- Exclusive

Because multiuser access to the database can increase or decrease depending on the application and data used, you should try to anticipate the number of concurrent users in the database and use this number as a guide for access modes.

If you schedule single-user access to perform large updates to the database, you can minimize locking resources and increase performance by specifying the exclusive write mode in the SQL SET TRANSACTION statement. The exclusive write mode does not incur the overhead of writing to the snapshot file or maintaining row-level locking information. Therefore, a transaction that uses the exclusive write mode can finish in less time and use less memory than the same transaction using another share mode and lock type.

If your application requires concurrent multiuser access, the access mode you choose will rely on the type of query being used. If the query requires only read access, use a read-only transaction. If the query updates the database, use the shared write mode where possible. The shared write mode promotes high concurrency. The protected write mode yields low concurrency and allows one single-update user in the specified table. Other users who attempt to write to the protected table will fail or wait, and must roll back. Users who request a read-only transaction are still allowed access to that table.

If your tasks require many concurrent update users in the table, use the shared mode in the SQL SET TRANSACTION statement. Other users may update the table. While the shared write mode increases the number of users allowed to update the table at the same time, it also increases lock contention for all users. Deadlocks may occur and force a user to redo work. Using the shared write mode increases transaction protection, but overall transaction turnaround may be slightly longer because users will wait longer for lock conflicts to be resolved.

Examples 3–20, 3–21, 3–24, 3–25, and 3–26 describe the reserving options and the level of compatibility with other transactions.

Example 3–20 Starting a Transaction in a Shared Read Reserving Option

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ WRITE RESERVING
cont> EMPLOYEES FOR SHARED READ;
```

A shared read request as shown in Example 3–20 grants all retrieval transactions access to the specified database resource. Other transactions that update a table reserved in this mode are also allowed access to the same resource. Shared readers and shared writers can access the same resource until the writer actually modifies the resource.

When a physical area is readied in shared read mode by one transaction and in shared write mode by the next transaction, the buffers belonging to the area are *not* flushed when the second transaction readies the area.

For all other mode conversions for physical areas, Oracle Rdb must flush the buffers belonging to the area before the second transaction readies the area. If the mode conversion is for a data area, Oracle Rdb first flushes the buffers belonging to the corresponding snapshot area, then flushes the buffers belonging to the data area, and finally demotes or promotes the locks, as appropriate.

When a table is reserved for shared write or shared read, the behavior is identical from a locking standpoint to the behavior when the reserving clause is not specified, except that:

- A reserving for shared read transaction always obtains locks in CR (concurrent read) mode, which prevents the transaction from writing to the table.
- Reserving for shared read and reserving for shared write transactions should never deadlock with transactions that specified the reserving for protected or reserving for exclusive share modes.

When a table is reserved, all the logical areas of the table are not readied prior to starting the transaction. Instead, only one logical area needs to be locked at the time the transaction starts to prevent a protected or exclusive transaction from succeeding. Subsequent logical areas are locked when accessed, as if no reserving clause had been specified. In the worst case, a reserving for shared read or reserving for shared write transaction would thus require one additional lock operation for each table accessed during the transaction.

Update transactions share access to the same table. There are several ways to minimize lock contention when you update rows. If you are updating rows whose indexes will not be modified, you can specify that dbkeys are valid until you issue a DISCONNECT statement. Use the SQL DBKEY SCOPE IS ATTACH clause when you first declare or invoke the database. This guarantees that the dbkey of each row does not change (in the case of deletions) until this user detaches from the database, usually with a DISCONNECT statement.

To minimize lock contention, start a read-only transaction by locating the rows you want to modify and their dbkeys. Issue a COMMIT statement. Then start a read/write transaction as shown in Example 3–21, look up the rows you want to modify by their dbkeys, and make the necessary modifications. This is a good method to use when you must make many updates in a concurrent environment.

Example 3–21 Starting a Transaction in a Shared Write Mode

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ WRITE RESERVING
cont> EMPLOYEES FOR SHARED WRITE;
```

The cost in I/O operations makes this method impractical for transactions in which only a few updates are required or for cases in which the indexes may change in the table being modified. Thus, if you are updating a row without changing the values of any columns that are part of an index, then you will have less lock contention if you retrieve the row by its dbkey.

For example, consider the EMPLOYEES table in the mf_personnel sample database. Assume that you retrieve a row for an employee and you want to change the employee's address. If your program retains the dbkey of the row, it could use the syntax shown in Example 3–22.

Example 3–22 Updating and Retrieving a Row by Dbkey

```
SQL> SELECT E.EMPLOYEE_ID
cont>  FROM EMPLOYEES E
cont>  WHERE E.RDB$DBKEY=<.saved dbkey>;
SQL>  UPDATE EMPLOYEES
cont>   SET ADDRESS_DATA_1 = <new address>
cont>   WHERE EMPLOYEE_ID=<.saved dbkey>;
```

In Example 3–22 only the data row is locked.

If the program fetches the row based on the value in the EMPLOYEE_ ID column (shown in Example 3–23), any index nodes for the index on EMPLOYEE_ID that must be traversed to reach the row are locked. The data row is also locked. If there is no index on EMPLOYEE_ID, then the entire table will be locked to find the row.

Example 3–23 Updating and Retrieving a Row by Column Value

```
SQL> SELECT E.EMPLOYEE_ID
cont>  FROM EMPLOYEES E
cont>  WHERE E.EMPLOYEE_ID=<some ID number>;
SQL> UPDATE EMPLOYEES
cont>  SET ADDRESS_DATA_1 = <new address>
cont>  WHERE EMPLOYEE_ID=<some ID number>;
```

Thus, in Example 3–22, fetching by dbkey reduces possible locking contention. A lock conflict is less likely because no index is locked.

If you modify a row using a dbkey, and the columns you modify are used in an index, then the index must be updated. This means that index nodes will be locked. In this case, fetching by dbkey does not reduce the possibility of lock contention. The same locks are held regardless of whether you retrieve by dbkey or by column value.

Another way to reduce lock contention, using an SQL precompiled or module language program, is to declare the same database twice with two different aliases. In the DECLARE ALIAS statement, you must specify the DBKEY SCOPE IS ATTACH clause to ensure that the dbkeys are always pointing to the same rows even if some rows are deleted. This prevents Oracle Rdb from using these dbkeys again until the user who deleted the rows issues a DISCONNECT statement. This method is useful for each update transaction you perform, if you include the database once in a read/write and once in read-only transaction. You can find the dbkeys or perform some other retrieval from the read-only copy of the database and then update the read/write copy. This method uses a good deal of the system resources, but can dramatically reduce lock contention. See the DECLARE ALIAS statement in the *Oracle Rdb7 SQL Reference Manual* for more information.

Transactions performing protected read operations can share the EMPLOYEES table. No other update transactions are permitted access. Example 3–24 shows how to declare a protected read transaction.

Example 3–24 Starting a Transaction in a Protected Read Mode

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ WRITE RESERVING
cont> EMPLOYEES FOR PROTECTED READ;
```

An update transaction as shown in Example 3–25 can access the EMPLOYEES table and share access with other retrieval transactions. No concurrent update transactions are permitted.

Example 3–25 Starting a Transaction in a Protected Write Mode

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ WRITE RESERVING
cont> EMPLOYEES FOR PROTECTED WRITE;
```

The exclusive share mode as shown in Example 3–26 grants only one transaction access to database resources at a time. The exclusive read mode lets you only read data from the EMPLOYEES table, while the exclusive write mode lets you insert, update, or delete data in the EMPLOYEES table.

Example 3–26 Starting a Transaction in an Exclusive Read or Write Mode

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ WRITE RESERVING
cont> EMPLOYEES FOR EXCLUSIVE READ;
cont> COMMIT;
SQL>
SQL> SET TRANSACTION READ WRITE RESERVING
cont> EMPLOYEES FOR EXCLUSIVE WRITE;
```

To ensure maximum concurrent access, select the most compatible locks when you request a database resource. For example, for very short transactions that have contention problems on an index, using protected write may be faster than shared write, even if there are concurrent users. A compatible lock can coexist with other locks for the same database resources. Table 3–8 shows lock compatibility between a current transaction and access modes other transactions may specify.

Table 3–8 Lock Compatibility Between a Current Transaction and Access Modes Other Transactions Can Specify

Mode of Requested	Mode of Current Lock				
Lock	SR	SW	PR	PW	EX
SR	Yes	Yes	Yes	Yes	No
SW	Yes	Yes	No	No	No
PR	Yes	No	Yes	No	No
PW	Yes	No	No	No	No
EX	No	No	No	No	No

Key to lock modes:

SR—Shared Read. Sometimes referred to as Concurrent Read (CR).
SW—Shared Write. Sometimes referred to as Concurrent Write (CW).
PR—Protected Read
PW—Protected Write
EX—Exclusive
Yes—Locks are compatible
No—Locks are not compatible

You may want to prevent other transactions from retrieving or updating a table. Specifying exclusive mode in your SQL SET TRANSACTION statement causes Oracle Rdb to lock the entire table. If the table is in a mixed storage area, an exclusive access transaction can prevent another transaction from accessing the tables in that storage area under the following circumstances:

- Hashed indexes are defined on the tables.
- The other transaction (or transactions) is read/write.
- The other transaction (or transactions) uses a hashed index in its query.

This is because tables in the same storage area share the same system records. While Oracle Rdb updates the hash bucket for the table in exclusive mode, the system record is locked, which prevents access to the other table. You should place tables that require updates in exclusive mode in separate areas.

Follow these guidelines to allow other users access to tables and rows in the database and to ensure lock compatibility:

- Specify only those rows you need by restricting the record stream in a precise record selection expression.
- Include the fewest possible queries in a single transaction.

- Specify a compatible lock mode in the SQL SET TRANSACTION statement that lets other users access the same resources concurrently.

See Section 3.8.5 for more information on how Oracle Rdb protects a database resource through locking.

3.8.3.1 Incompatible Share Mode and Lock Type

Oracle Rdb considers the exclusive write share mode and the read-only transaction type that uses the default read lock type to be incompatible because no snapshot files are written by an exclusive write share mode transaction. Transactions started in exclusive write mode cause subsequent read-only transactions to wait and fail when the exclusive write transaction attempts a COMMIT or ROLLBACK operation. If the read-only transaction includes the WAIT qualifier, Oracle Rdb returns a message that indicates that a resource is locked, converts it to a NO_WAIT qualifier, and returns a second message that indicates that the storage area cannot be readied for snapshot files, as shown in Example 3–27.

Example 3–27 Error Message for Incompatible Share Mode and Lock Type

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ ONLY WAIT;
SQL> SELECT * FROM EMPLOYEES;
%RDB-E-LOCK_CONFLICT, request failed due to locked resource
-RDMS-F-CANTSNAP, can't ready storage area $DUA0:[ORION]EMPIDS_LOW.RDA;1
for snapshots
```

Therefore, the user whose current transaction is the read-only transaction must roll back.

If your transaction requires exclusive access to an area of the database, you should be aware of the results of the exclusive mode on other read-only transactions that attempt to access the snapshot file.

3.8.3.2 Carry-Over Locks and the [NO]WAIT Option

Carry-over lock optimization allows transactions to avoid some logical area and physical area lock request overhead at commit time. This section provides a general description of how carry-over lock optimization works, and then describes its particular effects on WAIT and NOWAIT transactions.

An area lock requested by a current transaction is called an **active** lock. At commit time, Oracle Rdb tries to avoid demoting area locks. Those area locks that are not demoted at commit time are called **carry-over** locks.

While attached to the database, a process can have some active locks (locks used by the current transaction) and some carry-over locks (locks requested in earlier transactions that have not been demoted). If a transaction needs a lock that it has currently marked as carry-over, it can reuse the lock by changing it to an active lock. Thus, the same lock can go from active to carry-over to active multiple times without paying the cost of lock request and demotion. This substantially reduces the number of lock requests if a process accesses the same set of areas repeatedly.

WAIT Transactions

Whenever a WAIT transaction requests an area lock, Oracle Rdb must distinguish between the following two cases in which process A has a lock on area X and process B wants to access the same area:

- If the lock that process A has on area X is a carry-over lock, A gives up the lock on demand, process B gets it, and B continues to process.
- If the lock that process A has on area X is an active lock, A cannot give up the lock before its transaction has completed. In this case, Oracle Rdb sets a flag to indicate that this lock must be demoted when A's transaction commits, so that B can acquire it. Because process B cannot get the lock on demand, B must wait.

For WAIT transactions, the reduced number of locks associated with carry-over lock optimization can result in an increase in blocking ASTs. You can see an increase in blocking ASTs by using the various Performance Monitor Locking screens.

Carry-over lock optimization works well when applications are designed so that each transaction accesses its own set of data; that is, transactions do not randomly access data in all partitions, thereby increasing contention. For example, consider the EMPLOYEE_ID column, which partitions the EMPLOYEES table to three areas. Applications that access the EMPLOYEES table should be designed so that transactions access a particular area or set of areas instead of randomly selecting any area. Furthermore, carry-over lock optimization works best if transactions repeatedly access the same area or set of areas. The partitioning and placement features available in Oracle Rdb should help in this regard.

NOWAIT Transactions

NOWAIT transactions do not wait for locks. If a lock requested by a NOWAIT transaction cannot be granted immediately, Oracle Rdb issues an error message and the transaction aborts. As part of carry-over lock optimization, a NOWAIT transaction requests, acquires, and holds a NOWAIT lock. This signals other processes accessing the database that a NOWAIT transaction exists and results in the release of all carry-over locks. If carry-over locks were not released, a NOWAIT transaction could not access an area held by a WAIT transaction's carry-over lock until the WAIT transaction's process detached from the database.

However, a NOWAIT transaction can experience a delay in acquiring the NOWAIT lock if another transaction is holding the lock. This can result in the following Performance Monitor Stall message:

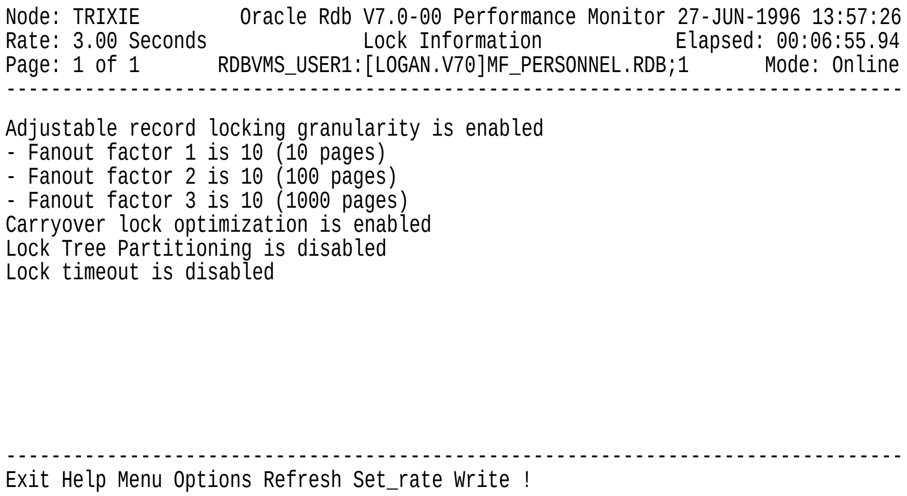
```
waiting for NOWAIT signal (CW)
```

If NOWAIT transactions are noticeably slow in executing, you can disable carry-over lock optimization by using the CARRY OVER LOCKS ARE [ENABLED | DISABLED] clause with either the SQL CREATE DATABASE or SQL ALTER DATABASE statements. By default, carry-over locks are enabled. Example 3–28 shows how to disable carry-over locks that have been enabled by default.

Example 3–28 Disabling Carry-Over Locks

```
SQL> ALTER DATABASE FILENAME test1
cont> CARRY OVER LOCKS ARE DISABLED;
```

You can determine if carry-over locks are enabled or disabled by examining the Performance Monitor Lock Information screen. For example:



```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 27-JUN-1996 13:57:26
Rate: 3.00 Seconds             Lock Information           Elapsed: 00:06:55.94
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------

Adjustable record locking granularity is enabled
- Fanout factor 1 is 10 (10 pages)
- Fanout factor 2 is 10 (100 pages)
- Fanout factor 3 is 10 (1000 pages)
Carryover lock optimization is enabled
Lock Tree Partitioning is disabled
Lock timeout is disabled






--------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
```

If you do disable carry-over lock optimization, you may see some degradation in performance. This is because every transaction will acquire and release area and top level ALG locks instead of trying to hold them across multiple transactions.

3.8.3.3 Optimizing Update Carry-Over Locks for Tables

Under normal circumstances, a process may start an update transaction that reads a row first, and then updates it later. In this case, Oracle Rdb may first grant the process a logical area lock in CR (concurrent read) mode, then upgrade the lock to CU (concurrent update) mode later. If the process starts another update transaction, the same situation can occur. Many lock operations can be required to transition from CR to CU mode and back again.

You can enable update carry-over locks at the table level for a process by defining the RDMS$AUTO_READY logical name or the RDB_AUTO_READY configuration parameter for the process. If the RDMS$AUTO_READY logical name or RDB_AUTO_READY configuration parameter is defined, when the process requests a logical area lock in CR mode, it will obtain that lock in CU mode if it is already holding a carry-over lock on the logical area in CU mode. This optimization of carry-over locks for logical areas reduces the locking overhead for most update transactions.

OpenVMS VAX OpenVMS Alpha

Example 3–29 shows how to enable update carry-over locking at a table level for a process. Note that carry-over locking must be enabled when you define the RDMS$AUTO_READY logical name to 1, otherwise defining the logical name will not enable update carry-over locks at the table level.

Example 3–29 Enabling Update Carry-Over Locking at a Table Level for a Process

```
$ DEFINE RDMS$AUTO_READY 1
```

To disable update carry-over locking at a table level for a process, deassign the RDMS$AUTO_READY logical name. ♦

A process benefits from update carry-over lock optimization for tables only if the process is updating the database. The benefit of enabling update carry-over locks at a table level for a process is that when the process starts a new update transaction on a table for which it holds a carry-over lock in CU mode for the logical area, it will get the logical area lock in CU mode. Getting the logical area lock in CU mode at the start of the transaction means the process avoids the locking overhead of first downgrading and then upgrading the lock later. If the process does not perform any updates, getting the logical area locks in the lower (CR) mode is sufficient.

The RDMS$AUTO_READY logical name and RDB_AUTO_READY configuration parameter should be used in high volume, update-intensive, transaction processing environments when the Performance Monitor indicates that a large number of lock conversions are taking place for each transaction.

A process that has enabled update carry-over locking at the table level can cause concurrency problems if the process reserves tables in PROTECTED READ or PROTECTED WRITE modes, or if it performs sequential scans of tables.

3.8.3.4 Explanation of “Lock Conflict on Freeze Lock” Errors

In some circumstances, it is possible for a NOWAIT transaction to receive a “lock conflict on freeze lock” error message. Consider the following potentially dangerous scenario:

Process A	Process B
Holds lock on resource 1.	
	Requests lock on resource 1 and is waiting.
Modifies resource 1.	
Process A is terminated prematurely.	

```
Operating system releases all locks
held by A.
                                          B is granted the lock on resource 1
                                          and may see uncommitted changes
                                          made by A.
```

When Oracle Rdb is running in a cluster, it is necessary to ensure that termination of a process (Process A) does not result in granting locks to other application processes (Process B) before the database monitor has detected the error and recovered the transactions for the failed process (if necessary).

The “freeze protocol” is used to ensure that any new lock request by another application process is not granted until the monitor has had a chance to do any cleanup (recovery) operations that are necessary for the terminated process. In the example, the freeze protocol ensures that Process B’s lock request is not granted until recovery operations for the terminated process (Process A) have been completed.

Here is how the “freeze protocol” algorithm works. Note that this explanation omits the intricate details to simplify the explanation.

When each user process attaches to the database, it gets the FREEZE lock in concurrent write (CW) mode.

Whenever there is a potential need to recover the database (when, for example, a process aborts abnormally or a node in the cluster fails), the FREEZE lock is requested by the database recovery (DBR) process in protected read (PR) mode. Because the DBR process requests the FREEZE lock in an incompatible mode, blocking ASTs are generated for every user process holding the FREEZE lock in CW mode. Every process gives up the FREEZE lock, then the DBR process acquires the FREEZE lock in PR mode and begins recovery operations.

Whenever a user process is granted a lock request for a resource, Oracle Rdb also checks whether the FREEZE lock is held by that process in CW mode. If the process does not have the FREEZE lock in CW mode, then Oracle Rdb knows that the monitor is trying to recover the database and return it to a consistent state. In this situation, the process that was granted the lock request to the resource needs to give it up so the DBR process can obtain the lock to the resource and recover the database.

Oracle Rdb releases the lock that the process was granted to the resource and (if the process’ lock request was a WAIT request) requests the FREEZE lock in CW mode for the process. After this is granted, Oracle Rdb reissues the process’ original lock request to the resource.

If the process' lock request was a NOWAIT lock request, Oracle Rdb flags the "lock conflict on freeze" message to indicate that the lock request cannot be granted because (potentially) recovery is in progress. Any of the following events (and perhaps others) could cause the "lock conflict on freeze" error message to occur:

- The lock request is a NOWAIT request and recovery is taking place (as shown by DBR processes in the monitor log).
- A node failed while users were accessing the database.
- The monitor log has logged events such as: "cluster recovery completed successfully" or "received request from remote node to join . . . ". This indicates that new database activity is occurring on a node, or there are no more users on a node accessing a database. When these events happen, the monitor has to ensure that this transition did not leave the database in an inconsistent state.

3.8.3.5 Batch-Update Transactions

Batch-update transactions can be useful if you want to reduce overhead in large, initial-load operations. However, Oracle Corporation recommends that you use the read/write exclusive mode for these initial load operations because batch-update transactions offer some risks if problems develop. In batch-update transactions, no .ruj file is created, so if the transaction fails, the operation cannot be rolled back. Then the database becomes corrupt, and you must either restore the database from the most recent backup file or start over. Be sure to back up your database before you attempt to use batch-update transactions for initial load operations if you use this method for loading tables. When you use the exclusive share mode, an .ruj file is created, so you can roll back the transaction if the transaction fails. See the *Oracle Rdb7 Guide to SQL Programming* for more information on how to use batch-update transactions.

3.8.3.6 Update Locking for Cursors

When you use a cursor to select a record stream to read and possibly update rows, Oracle Rdb cannot know if every fetched row will be updated. The default Oracle Rdb behavior is to lock the specified rows for READ and then upgrade the lock to WRITE if a row is updated. With the current version of Oracle Rdb, if you know in advance that all or most of the fetched rows will be updated, you can apply more restrictive locking during the initial read operation. This results in lower locking overhead and reduced deadlocks.

To define an update cursor, use the UPDATE ONLY clause of the DECLARE CURSOR statement. The generalized statement is shown in the following example; for complete syntax details, see the *Oracle Rdb7 SQL Reference Manual*.

```
DECLARE cursor_name UPDATE ONLY CURSOR FOR select-expr
```

3.8.4 Transaction Scope

All database activity is controlled within transactions. When you begin a transaction, you specify the kind of database access other users can have to data while your transaction is active. You can restrict all access to the tables you are using or you can allow other users to share the same data. Whatever the access mode your transaction specifies, it, along with other active transactions, accumulates an increasing set of restrictions on the data resource. It is common to wait for a row to be released by another transaction before you can retrieve the row. Because Oracle Rdb lets the user fetch one row or several rows in a record stream, the more rows you retrieve, particularly for an update, the greater the chance of conflict with another user. When you terminate your transaction, you release all locks on the rows you retrieved.

To reduce potential lock conflicts with other users of the database, you should keep your transactions as short as possible while allowing the greatest degree of data sharing within your transaction. If you need to retrieve a row to display certain columns on a user's terminal, you should specify a share mode that allows other users to read the same rows. You can share access to the rows you need to retrieve if you specify READ ONLY in your SQL SET TRANSACTION statement. Avoid writing transactions that request terminal input from the user within the transaction that is holding locks on rows.

If you are updating the database, specify shared write in your read/write transaction. As soon as you modify the row, terminate your transaction. In this way, you surrender the lock resource to the system and allow another user to read or update that row if needed. If you retrieve a row in read-only mode and then decide to update that row, you can start another transaction, specifying read/write to modify the row in the database. If you delete the row, the dbkey scope determines when the dbkey of the deleted row can be used again by Oracle Rdb. If you specify DBKEY SCOPE IS TRANSACTION, the default, Oracle Rdb cannot use the dbkey of a deleted row again to store another row until the transaction that deleted the original row completes with a COMMIT statement. If you specify DBKEY SCOPE IS ATTACH, Oracle Rdb cannot use the dbkey of a deleted row again to store another row until the user who deleted the original row detaches from the database with a DISCONNECT statement.

Dbkey scope is determined in the following manner:

- If all active users use the DBKEY SCOPE IS TRANSACTION clause, the dbkey scope is TRANSACTION for all users.

- If just one user attaches to the database using the DBKEY SCOPE IS ATTACH clause, then the dbkey scope is ATTACH for all users.

Keeping the transaction as short as possible, that is, minimizing the database activity within the transaction, helps to maintain concurrent access at an optimal level. In the event of a system failure, short update transactions permit faster recovery and make the database current up to the last committed transaction. The longer the uncommitted transaction, the more work there is to do after a system failure. You will have to reenter what was rolled back. Because the size of your .ruj file increases with the scope of your update transaction, you also risk an increased demand for disk space for the .ruj files.

Table 3–9 summarizes the impact of the transaction scope on database users, files, and performance.

Table 3–9 Effect of Long and Short Database Transactions

Extent of Update Transaction	Impact	
Long with exclusive share mode	Advantages:	Updates quickly Eliminates access contention Uses fewest locks
	Trade-offs:	Reduces concurrency Slower recovery from system failure
Long with shared share mode	Advantages:	Greater concurrency
	Trade-offs:	Requires more locks Updates are longer because of high contention
Short	Advantages:	Database is current Reduces .ruj file size and storage requirements Faster recovery from system failure Increases concurrency
	Trade-offs:	Requires more transactions and higher lock manager overhead

3.8.5 Adjustable Lock Granularity

Adjustable lock granularity (ALG) is the mechanism Oracle Rdb uses to maintain as few locks as possible during a transaction, while still ensuring maximum concurrent access to pages within a logical area. ALG attempts to lock groups of rows (index nodes or data rows), anticipating that additional rows, which may need to be accessed later in the same transaction, are included in the locked group.

ALG is based on a logical, inverted tree structure that maintains locks on database resources at a variety of levels. Oracle Rdb defaults to three page range levels by default with a fanout factor of 10 at each level. You can control the lock granularity further by specifying the number of page levels with the COUNT parameter of the ADJUSTABLE LOCK GRANULARITY clause. The value of the COUNT parameter can range from 1 to 8. A value of 8 is interpreted as follows:

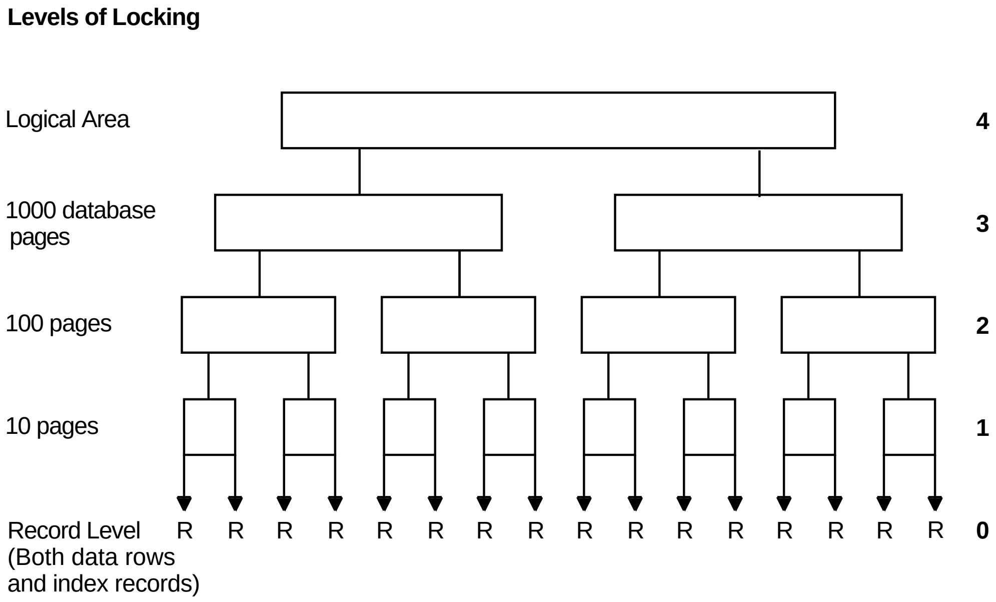
- Level 10, the top of the inverted tree (the root), includes all the rows found in a logical area.
- Level 9 includes the rows found on 1,000,000,000 consecutive pages within the logical area.
- Level 8 includes the rows found on 100,000,000 consecutive pages within the logical area.
- Level 7 includes the rows found on 10,000,000 consecutive pages within the logical area.
- Level 6 includes the rows found on 1,000,000 consecutive pages within the logical area.
- Level 5 includes the rows found on 100,000 consecutive pages within the logical area.
- Level 4 includes the rows found on 10,000 consecutive pages within the logical area.
- Level 3 includes the rows found on 1000 consecutive pages within the logical area.
- Level 2 includes the rows found on 100 consecutive pages within the logical area.
- Level 1 includes the rows found on 10 consecutive pages within the logical area.
- Level 0, the bottom of the inverted tree (the leaves), is the row level.

If your database has high page contention (many users accessing the same area simultaneously), consider specifying a lower COUNT value. If your database has few users who are performing queries that access many widely dispersed rows, specifying a higher COUNT value may result in less locking.

If you specify ADJUSTABLE LOCK GRANULARITY IS DISABLED, Oracle Rdb requests a lock for each database row requested. Oracle Rdb recommends that you start with adjustable lock granularity enabled, using the default COUNT value and then, if CPU time is a problem, adjust the count or disable the adjustable lock granularity and determine if this improves your performance.

Figure 3–1 illustrates the default ALG tree structure.

Figure 3–1 Adjustable Lock Levels


Levels of Locking
Logical Area
4
1000 database pages
3
100 pages
2
10 pages
1
Record Level (Both data rows and index records)
R R R R R R R R R R R R R R R R
0

NU–2172A–RA

If you are using a default value of 3 for the page level, all transactions initially request a strong lock at level 4 (the logical area). If there is no contention (no other transactions want to access rows in the same logical area), a transaction needs only one lock for all the rows in that logical area. This is because a strong lock at a higher level implicitly locks all objects at the lower levels that it dominates.

If another transaction needs to access rows from the same logical area, the first transaction’s strong lock at level 4 is de-escalated to a weak lock, and a strong lock is requested at level 3 for the appropriate page range. The second transaction also acquires a weak lock at level 4 and a strong level 3 lock on the appropriate page range. If there is conflict at level 3, the first transaction’s strong level 3 lock is de-escalated to a weak level 3 lock, and a strong level 2 lock is requested for the appropriate page range. The second transaction follows the same procedure. This paired acquisition and de-escalation of locks continues towards the leaves of the ALG tree until there is no contention. If the two transactions attempt to access the same row, conflict eventually occurs at the row level. In this case, the second transaction must wait until the first transaction finishes and releases all its locks.

The number of locks required depends on the number of transactions attempting to access rows in a logical area, and how many conflicts have been encountered in the ALG tree. By default, in the worst case, a transaction would need weak locks at levels 4, 3, 2, and 1 before finally acquiring a strong row lock at level 0, thus requiring a total of 5 locks to access a single row. This worst case would arise *only* if there is high contention on nearby rows.

This mechanism is useful when a number of transactions need to access different groups of pages concurrently. Enabling ALG can reduce lock activity if applications are designed so that transactions access data rows located on nonconflicting page ranges.

As a general rule, you should enable ALG if transactions need to access many rows stored in close proximity to each other. Because ALG takes out the highest level lock possible for a transaction, rows can be accessed with the minimum number of locks for each transaction.

You should disable ALG if transactions need to access only a few rows from a logical area. In this case, it is more efficient to allow Oracle Rdb to lock the lowest level immediately. Remember, if ALG is enabled and there is contention for pages at a low level, ALG must take out locks at all the intermediate levels of the ALG tree even if the transaction needs to access only a single row.

If a transaction needs to access millions of rows, you should set the transaction to lock the table in a protected mode. The protected mode lock saves the overhead of performing implicit or explicit locking for each accessed row. This can save a considerable amount of virtual memory. In other words, if a transaction accesses a majority of the rows in a table, you should reserve the table in a protected mode.

Note

> ALG is enabled or disabled for the entire database, not for specific transactions. You should determine the general characteristics of all the transactions running on your database before you decide to enable or disable ALG.

When you create a database, ALG is enabled by default. To disable ALG, use the ADJUSTABLE LOCK GRANULARITY IS DISABLED clause of the SQL ALTER DATABASE statement. See Section 8.4 for more information on when to enable and disable adjustable lock granularity.

The Performance Monitor Stall Messages screen can indicate where lock contention exists in the ALG tree. For example, the stall message `waiting for record 5:0:-4` indicates the database system is attempting to acquire an ALG lock and should be interpreted as follows:

- The first number (in this case, 5) indicates the logical area number.
- The second number (in this case, 0) indicates the starting page of the page range that is locked by this ALG lock.
- The negative number (in this case, –4) indicates the level in the ALG tree, and can range from –10 for a logical area lock, to –1 for a lock on a range of ten pages.

Refer to Section 3.2.1.3 for additional information on how to use and read the Stall Messages screen.

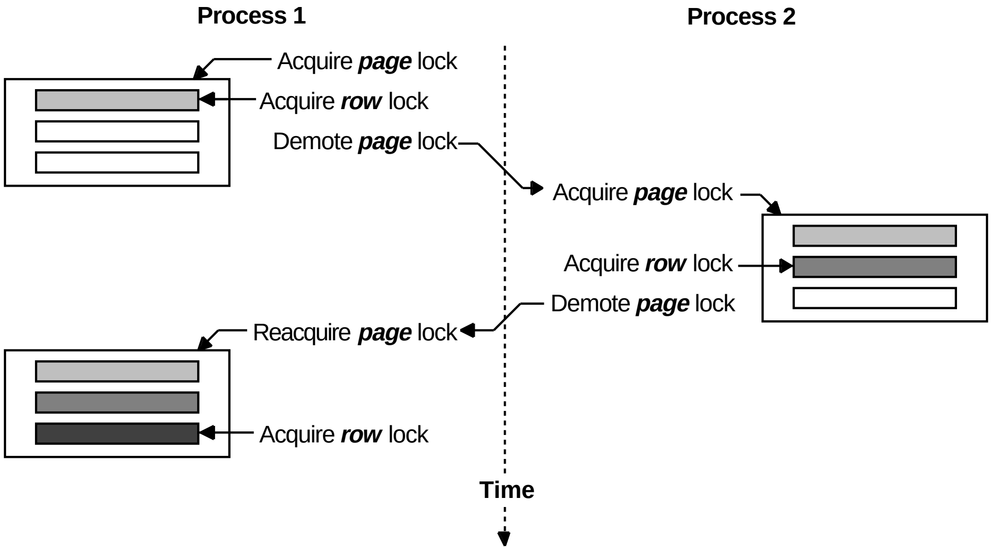
3.8.6 Selecting Page-Level or Row-Level Locking

Oracle Rdb uses row-level locks to provide logical consistency and page-level locks to provide mutual exclusion. The page-level and row-level locking mechanisms operate independently of each other.

Figure 3–2 shows two processes accessing different records on a data page. The row-level locks provide logical consistency, and the page-level locks ensure that the changes are ordered one after the other.

In Figure 3–2, Process 1 acquires the page lock and a row lock to one of the rows on the page. When Process 2 wants to access a different row on the page, Process 1 gives up the page lock, and Process 2 acquires the page lock and the row lock for the row it wants to access. When Process 1 wants to access another row on the page, Process 2 gives up the page lock, and Process 1 acquires the page lock and the row lock for the row it wants to access. After a process acquires a row lock, it holds that row lock until the end of its transaction (the process continues to hold the row lock even when another process has a page lock for the page). Oracle Rdb allows a process to *update* a row on a page only when the process has the page lock for that page (assuming that the transactions are executing at isolation level SERIALIZABLE). Only one process has the page lock for a page at a time, so only one process at a time can update rows on a page.

Figure 3–2 Page-Level and Row-Level Locking When Two Processes Access Different Rows on a Data Page



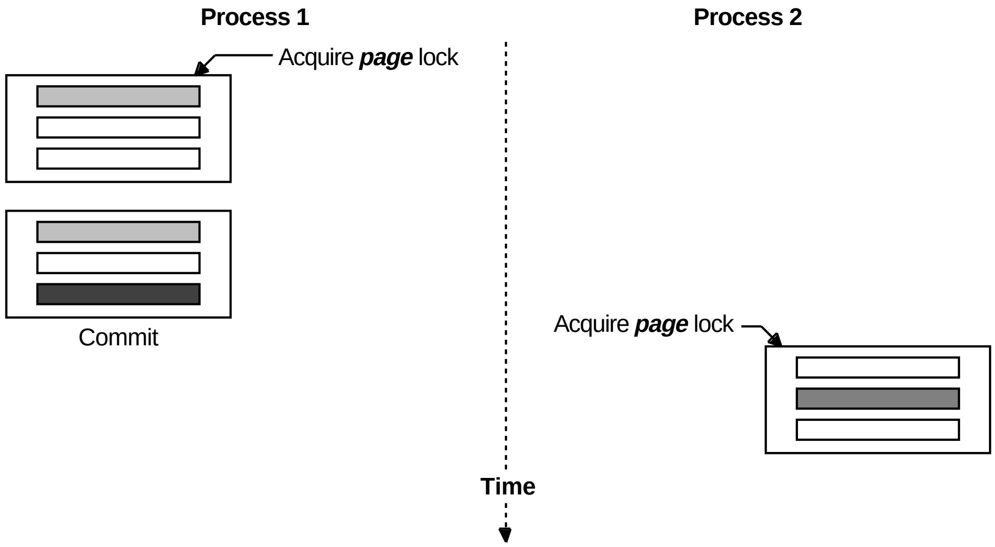
NU−2958A−RA

You can specify that *only* page-level locking be used for one or more storage areas or for all the storage areas in the database by using the LOCKING IS PAGE LEVEL clause of the SQL CREATE DATABASE and ALTER DATABASE statements. The *Oracle Rdb7 SQL Reference Manual* describes the syntax for these statements. When page-level locking is enabled for a storage area, transactions accessing the storage area hold only page-level locks and do not request any row-level locks. The page-level locks provide both logical consistency and mutual exclusion.

When page-level locking is enabled for a storage area, a process that accesses rows from that area is granted a page lock for each page accessed, but the process does not need to obtain any row-level locks for any of the rows accessed on the page. This reduces locking overhead.

When the LOCKING IS PAGE LEVEL clause is specified, page locks are held until the end of the transaction, which may cause other transactions that need to access rows on the page to be blocked. Thus, when you reduce lock operations by specifying the LOCKING IS PAGE LEVEL clause, you may also reduce concurrency. In Figure 3–3, page-level locking has been enabled and the two processes shown in Figure 3–2 are accessing the same rows on the same data page. When page-level locking is enabled, a process that gets a page lock is allowed to hold the lock until it completes the current transaction with a COMMIT or ROLLBACK statement. All other users are blocked from the page until the process that holds the page lock completes the current transaction. In Figure 3–3, a smaller number of locks are used than in Figure 3–2, but concurrency is also reduced because each process blocks the other when it has the page lock. If more processes were trying to access the page, the stall times for all these processes would increase.

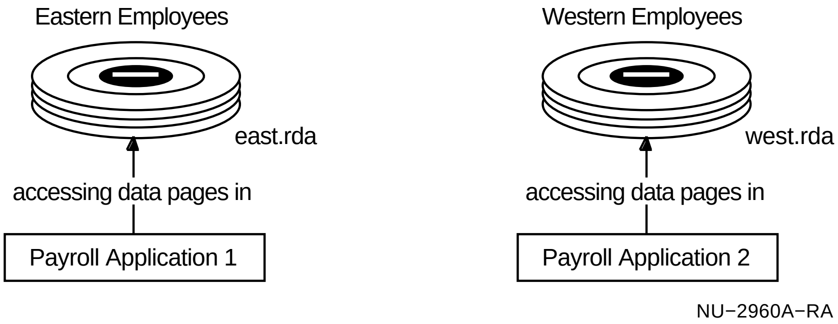
Figure 3–3 Reduced Concurrency When Page Locking Is Enabled and Two Processes Access Different Rows on a Data Page



NU−2959A−RA

Enabling page-level locking is most likely to benefit a partitioned application, in which individual application processes do not access the same data pages at the same time. Figure 3–4 depicts a payroll application in which information on employees is partitioned based on an employee's geographical location. That is, information on eastern employees is stored in the EAST storage area, and information on western employees in the WEST storage area. When separate application processes are used to update the employee information for each geographical location, the same data page should be requested seldom, if ever, by more than one application process at the same time. In such situations, you can enable page-level locking to remove the overhead of row locks, thereby improving performance without reducing concurrency.

Figure 3–4 Partitioned Application Likely to Benefit from Page Locking



Enabling page-level locking may also benefit applications with the following characteristics:

- Contention for resources is limited

  After page-level locking is enabled, a process will be able to quickly obtain a single page lock for a page and all the rows on the page.

- Transactions are short in duration

  After page locking-level is enabled, processes hold page locks until the end of a transaction. When transactions are short, a conflicting process has to wait only a short time if it requests a page already locked by another process.

Enabling page-level locking could also benefit an application that accesses a storage area in which all the rows on each data page are clustered on that page because they have the same value for a hashed index key. For example, if a group of rows is clustered (hashed) to the same page, the application can fetch the page and read or modify all the rows using only one page lock, instead of getting individual row locks for each row. However, the application is likely to benefit from enabling page-level locking only when no other processes or applications are attempting to access the same data pages at the same time as the application. If more than one process tries to access a data page at the same time in a storage area with page locking enabled, each process should have *very* short transactions; otherwise, each process will stall regularly waiting for page locks, thereby decreasing performance.

You should use the LOCKING IS PAGE LEVEL option carefully, only when the implications of enabled page-level locking have been considered. When the LOCKING IS PAGE LEVEL option is used for a storage area, page-level locking is enabled for the entire storage area. Most applications operate on logical entities such as tables, and when multiple tables exist within a storage area, special care must be taken when enabling page-level locking. The decision to enable page-level locking is easier if the storage area contains rows for only one table, and only one application accesses the table at a time. Enabling the LOCKING IS PAGE LEVEL option for a storage area is more likely to result in higher contention, longer delays, and more deadlocks as the number of tables in the area and the number of applications accessing the area at the same time increases.

The default is LOCKING IS ROW LEVEL. When the LOCKING IS ROW LEVEL option is in effect for one or more storage areas, a transaction accessing the storage area will use both row and page locks. In general, the LOCKING IS ROW LEVEL option is appropriate for most transactions, especially those that lock many rows in one or more storage areas and that are long in duration.

The LOCKING IS PAGE LEVEL and LOCKING IS ROW LEVEL clauses can be specified with the SQL CREATE DATABASE and ALTER DATABASE statements. These clauses can be specified at the storage area level (using the CREATE STORAGE AREA or ALTER STORAGE AREA clauses) or the database level.

Example 3–30 shows how to enable page-level locking for one storage area (JOBS) in a database.

Example 3–30 Modifying the LOCKING IS Setting for a Storage Area

```
SQL> -- Change the setting of the JOBS storage area from the default
SQL> -- setting of LOCKING IS ROW LEVEL to LOCKING IS PAGE LEVEL:
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ALTER STORAGE AREA JOBS
cont>    LOCKING IS PAGE LEVEL;
SQL>
```

Example 3–30 shows how the LOCKING IS clause can be used as part of the ALTER STORAGE AREA clause to specify page or row-level locking for a single storage area. To specify page or row-level locking for more than one storage area, specify the LOCKING IS clause as part of the ALTER STORAGE AREA clause for each storage area. You can also use the LOCKING IS clause without the ALTER STORAGE AREA clause as a convenient shorthand method for enabling page or row locking for all the storage areas in a database. Example 3–31 shows how to enable page-level locking for all the storage areas in the mf_personnel database.

Example 3–31 Enabling Page Locking for All the Storage Areas in a Database

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> LOCKING IS PAGE LEVEL;
```

Remember that the LOCKING IS PAGE LEVEL and LOCKING IS ROW LEVEL settings are storage area attributes (because they can be changed on a per storage area basis), not database-wide attributes.

The Performance Monitor provides the lock level setting for each storage area. Select the Storage Area Information screen from the Database Parameter Information submenu to display storage area characteristics, including the lock level settings. Section 4.2.1.3 shows an example of a Storage Area Information screen.

You can set page-level or row-level locking only when the database is off line (when other users are not attached to the database).

The following restrictions apply to enabling page-level locking:

- Page-level locking is illegal for single-file databases. You will receive an RDB$_BAD_DPB_CONTENT error if you attempt to enable page-level locking for a single-file database.
- Page-level locking is never applied to the RDB$SYSTEM storage area by the SQL CREATE DATABASE or ALTER DATABASE statements because the locking protocol may stall metadata users.
- Page-level locking cannot be specified explicitly for the RDB$SYSTEM storage area. If you attempt to do this, Oracle Rdb issues an RDB$_BAD_DPB_CONTENT error that explains page-level locking is not allowed for the RDB$SYSTEM storage area.

Although the proper use of the LOCKING IS PAGE LEVEL feature results in fewer locks being requested by a transaction, quotas should not be adjusted based on use of this feature.

3.8.7 Recoverable Latches

Recoverable latches are Oracle Rdb locks used for node-specific locking operations where exclusive locks for low contention resources are required. A recoverable latch performs the same function as an Oracle Rdb lock, but the advantage of the recoverable latch is that it can be locked or unlocked with fewer instructions than the Oracle Rdb lock.

Recoverable latches maintain the information needed by the database recovery (DBR) process to recover a database that fails.

Note that no syntax is required to enable recoverable latches. Oracle Rdb uses recoverable latches automatically in databases with global buffers enabled. This feature improves performance for any database that has global buffers enabled.

3.8.8 Read-Only Storage Areas

For storage areas with stable data that are not expected to grow, you can modify the access to the tables in the storage area to be read-only. To do this, change the read attribute for the storage area from read/write to read-only using an SQL ALTER DATABASE statement. This change eliminates the possibility of users taking out write locks in read/write transactions to read data from a table and getting page-level or row-level locks from other users performing similar tasks. For example, to change the DEPARTMENTS storage area to read-only, use the statement shown in Example 3–32.

Example 3–32 Changing Read/Write Status to Read-Only

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ALTER STORAGE AREA DEPARTMENTS READ ONLY;
```

A read-only storage area is a useful way to deal with stable data in your database. To add rows to a table stored in a read-only storage area, just change the read-only attribute back to read/write using the SQL ALTER DATABASE statement. See the *Oracle Rdb7 Guide to Database Maintenance* for developing backup and restore operation strategies for databases that contain read-only storage areas.

3.8.9 Using an RDB$SYSTEM Read-Only Storage Area

For databases with stable metadata and tables in the RDB$SYSTEM default storage area that are stable in size, you can also change the read/write attribute of the RDB$SYSTEM default storage area to read-only. This change eliminates the page and row locking in this read-only storage area. Note that you cannot change any metadata other than the cardinalities of logical areas using the RMU Collect Optimizer_Statistics command after the RDB$SYSTEM storage area is set to read-only. You must set the read-only attribute back to read/write for the RDB$SYSTEM storage area to make a change in the metadata, such as changing the protection for users accessing the database.

Note

When you set a storage area to read-only, objects in that area cannot be readied in update mode, including any indexes created in the area. This means that if an index that is stored in a read-only area is readied in update mode, the operation will fail. For example, if you set the RDB$SYSTEM storage area to read-only, the following query fails:

```
SQL> UPDATE EMPLOYEES
cont> SET MIDDLE_INITIAL = NULL
cont> WHERE EMPLOYEE_ID > '00200';
%RDMS-F-READONLY, data in a read-only storage area may not be accessed
for update
```

Because the query specifies a range retrieval, the optimizer used the sorted index on EMPLOYEE_ID to retrieve the rows to be updated. This sorted index is located in the RDB$SYSTEM area. When the query executes, Oracle Rdb tries to ready the index area (RDB$SYSTEM) in update mode, and the operation fails.

Therefore, if you set the RDB$SYSTEM storage area to read-only, ensure that RDB$SYSTEM does not contain any indexes required by update operations.

You can change the RDB$SYSTEM storage area to read-only as shown in Example 3–33.

Example 3–33 Changing the Read/Write Status of the RDB$SYSTEM Storage Area to Read-Only

```
SQL> ALTER DATABASE FILENAME mf_personnel READ ONLY;
```

When the RDB$SYSTEM storage area is set to read-only, automatic updates to table and index cardinalities are disabled. This has the side effects of stabilizing manual changes to the cardinalities (number of rows) in the metadata tables and indexes that influence the query optimizer, and eliminating the I/O operations associated with the cardinality update.

To update the cardinality of all tables and indexes in the RDB$SYSTEM storage area while it is read-only, use the RMU Collect Optimizer_Statistics command.

Note

You must have snapshots enabled and allowed to use the RMU Collect Optimizer_Statistics command. Because cardinality table counts are made by accessing each row of each table, locking in any other manner would cause the entire database to become locked out.

For example, suppose you want to update the cardinalities of the tables and indexes in the RDB$SYSTEM storage area to reflect new cardinalities for tables and indexes that have had new rows added since you set the RDB$SYSTEM storage area to read-only. You may want to do this to permit the query optimizer to devise the best query strategy based on the most current values for all table and index cardinalities stored in the metadata in this storage area. You would update the metadata as shown in Example 3–34.

Example 3–34 Updating Table and Index Cardinalities in RDB$SYSTEM

```
$ RMU/COLLECT OPTIMIZER_STATISTICS/STATISTICS=CARDINALITY mf_personnel
$ rmu -collect optimizer_statistics -statistics=cardinality mf_personnel
```

This command updates the cardinality values in the RDB$CARDINALITY column of the system tables RDB$RELATIONS and RDB$INDICES.

Note

Dynamic optimization attempts to approach optimal query performance by relying heavily on the statistics collected during query execution. If index and table cardinalities are artificially shifted away from their correct values, they will contradict the dynamically collected statistics and cause quite unpredictable selection of strategies and sequence of their evaluation. Hence any adjustment of cardinalities will most probably result in poorer query optimizer performance.

3.9 Index Retrieval

Index retrieval is one of the two methods Oracle Rdb uses for accessing rows; the other method is sequential access.

There are three types of index access:

- Direct index access. Oracle Rdb uses a unique index key to locate data and access it directly from the table. In this case, the key matches only one row.
- Retrieval by index. Oracle Rdb uses a partial index key or a duplicate to locate data. Oracle Rdb scans through the index until it finds all rows that match the key at the bottom of the range of a search (the low Ikey) and all rows that match the key at the top of the range (high Ikey). For both low Ikeys and high Ikeys, Oracle Rdb uses the dbkey to read the actual row. For more information on low Ikeys and high Ikeys, see Appendix C.
- Index-only retrieval. Oracle Rdb uses the index to find the row entry. In this case, the data is part of the index key; hence, Oracle Rdb returns the data from the index, rather than using the dbkey to read a row.

In terms of cost in the number of I/O operations, index-only retrieval is the most efficient. Retrieval by index requires the most CPU overhead. Sequential access works best for small tables.

Indexes are used not only for data retrieval, but also for update operations.

An update transaction can update other columns for which no indexes are defined. You can use an indexed column to locate a row and update another nonindexed column without incurring the overhead required to update the index.

A transaction that updates an indexed column does the following:

- Locks the index node that refers to the updated row.
- Locks any index nodes that must be split when the transaction adds more rows to the table. The transaction, then, also updates the index itself.
- Prevents any read/write transactions from accessing the database rows through those index nodes.

If your transaction uses the dbkey itself, rather than allowing Oracle Rdb to find the dbkey for you using the index, your program can access the row directly without using the index. Refer to the *Oracle Rdb7 SQL Reference Manual* for more information about dbkeys.

3.9.1 Types of Indexes

Oracle Rdb uses two types of indexes:

- Sorted indexes

  A sorted index is generally the best overall access method. It offers good performance for exact matches, determining the existence of a row, retrieving ranges of values, and retrieving partial keys; and when clustered (clustered B-tree) performs very well for sorted retrievals. It works best with dynamic tables.

- Hashed indexes

  A hashed index performs best for exact match retrieval, especially when used in very large tables where the B-tree access method does not do so well due to the overall large size of the B-tree for tables of this size.

After you define a sorted index structure, Oracle Rdb can access the rows in a table by searching the index to find a dbkey and getting the information directly from the index key, or finding the row in the database by using the dbkey.

After you define a hashed index structure, Oracle Rdb can access rows in a table through direct access by searching the hash bucket for the search key and dbkeys and going directly to the relative data page number where the data row is located. Hashed indexes are best for random, direct access for exact matches of the search key.

When you use sorted index retrieval, be sure that adjustable lock granularity (ALG) is enabled. ALG greatly improves database concurrency by automatically adjusting the level of locking granularity downward from the highest level (the entire logical area, table, or index) to lower levels that encompass successively smaller groups of pages until the lowest level (an individual row) is reached when other retrieval or update transactions access index nodes or rows in the table. However, access by other transactions to those rows through the sorted index may encounter wait conditions because of locking at index nodes that have been updated and locked. See Section 3.9.6 for details on forming indexing strategies.

When you define sorted and hashed indexes for specific columns in the database, you give the optimizer the option to choose an access method to retrieve data with fewer disk accesses than may be required using a sequential search and retrieval. As a database designer, you must carefully identify those columns that can benefit from either type of index. The primary key is always a good candidate for an index, but certain read-only tasks may also benefit from indexes on other columns.

See Section 3.9.6 for more information about sorted indexes. See Section 3.9.7 for more information about hashed indexes.

3.9.2 Logical Area Names for Indexes

For a single-file database, Oracle Rdb stores all the index structures for a table in a single logical area of the .rdb file and treats the pages allocated to this logical area as data pages; that is, indexes contain the same data structure as any other data page.

For a multifile database, you can use the STORE clause to assign the storage area in which to store the sorted index structures for a table. Oracle Rdb stores each individual index structure for a table in individual logical areas of the .rda file and treats the pages allocated to this logical area as data pages.

Logical area names appear in the output from the RMU Analyze command and the RMU Dump command with the Larea=RDB$AIP qualifier and help to analyze the database structure.

Oracle Rdb uses the convention that indexes created explicitly or implicitly in the RDB$SYSTEM storage area all share the same logical area if they are for the same table. The name of the first index is assigned as the name of the logical area used by all indexes for the same table.

For example, consider this simple table and index definition:

```
SQL> CREATE TABLE TEST (I_COL integer, C_COL char);
SQL> CREATE INDEX IND_1 on T (I_COL);
SQL> CREATE INDEX IND_2 on T (C_COL);
```

Index IND_1 and IND_2 are both mapped by default to the RDB$SYSTEM storage area. There is one logical area created for the indexes of table TEST and it is named after the first created index (in this case IND_1) and contains information for all indexes of table TEST mapped to the RDB$SYSTEM storage area (in this case IND_1 and IND_2).

If the index IND_1 is subsequently dropped and re-created in another storage area, the database administrator will notice that the logical area IND_1 still exists in RDB$SYSTEM (physical area 1). The IND_1 name is retained because it still contains data for the IND_2 index. This may cause confusion because the logical area IND_1 now exists in multiple storage areas even though the index itself no longer resides in RDB$SYSTEM.

If later still, the index IND_2 is dropped and re-created in another storage area, the original IND_1 logical area in RDB$SYSTEM will be removed because it is unused.

If indexes are mapped to storage areas other than RDB$SYSTEM, they will always be assigned a separate and private logical area.

Note

Oracle Corporation recommends that all new databases be created as multifile databases and that all indexes and tables have storage maps. This will avoid this point of confusion as well as allow more flexibility for database physical restructuring.

This convention of sharing logical areas for indexes is part of the single-file database model and is retained for upward compatibility with older versions of Oracle Rdb. The benefit of this convention is to save space in single-file databases and for system tables that are always stored in the RDB$SYSTEM storage area.

3.9.3 Index Compression

You can specify that indexes should be compressed. In a **compressed index**, information within the index nodes is reduced in size so that data takes up less space. The four types of index compression are:

- Prefix and suffix compression, described in Section 3.9.3.1
- SIZE IS segment truncation, described in Section 3.9.3.2
- MAPPING VALUES compression, described in Section 3.9.3.3
- Run-length compression, described in Section 3.9.3.4

The benefits of index compression include:

- Much lower storage requirements for some applications
- Fewer I/O operations to retrieve data, because more user index nodes may be included in buffers
- More efficient index-only retrieval, because more data may reasonably be included in an index

3.9.3.1 Prefix and Suffix Compression

Prefix and suffix compression occur automatically for sorted indexes only. At the lowest level of the index, which points to the data rows (level 1 index nodes), only prefix compression takes place.

At higher levels of the index, both prefix and suffix compression occur. For these higher level index nodes, prefix compression consists of removing leading bytes that are identical between consecutive index keys from the front of the index key. Suffix compression consists of removing noninformative bytes from the back of the index key. See the description of Example 3–46 in Section 3.9.5.1 for more information on index key prefixes and suffixes.

3.9.3.2 SIZE IS Segment Truncation

Use the SIZE IS clause of the CREATE INDEX statement to specify that the “first n” characters of a certain key are to be used in the index. For example, to place an index on a 100-byte column that is generally unique in the first 20 bytes, you could specify that only the first 20 bytes be used in the index and save as much as 80 bytes per entry. You can specify segment truncation for sorted indexes only.

To create a SIZE IS compressed index for columns that use the CHAR or VARCHAR data types, use the SIZE IS clause of the SQL CREATE INDEX statement for the column or columns being indexed, as shown in Example 3–35.

Example 3–35 Setting SIZE IS Index Compression for a CHAR Data Type Column

```
SQL> CREATE INDEX EMP_LAST_NAME ON EMPLOYEES
cont>  (LAST_NAME SIZE IS 10)
cont>   TYPE IS SORTED;
```

You can specify the UNIQUE clause when you create a SIZE IS compressed index. However, if you specify the UNIQUE clause with a SIZE IS compressed index, truncating the index key values may make the key values not unique. In that case, the index definition or insert or update statements will fail. For example, if you define the index EMP_LAST_NAME index as unique and attempt to insert a row for an employee with the last name “Kilpatrick” and a row for an employee with the last name “Kilpatricks”, SQL returns the following error:

```
%RDB-E-NO_DUP, index field value already exists; duplicates not allowed for
EMP_LAST_NAME
```

3.9.3.3 MAPPING VALUES Compression

When you specify MAPPING VALUES compression, Oracle Rdb reduces the number of bits needed to store indexes of all numeric columns by translating the column values into a more compactly encoded form. You specify this type of compression with the MAPPING VALUES clause of the SQL CREATE INDEX statement. Note that MAPPING VALUES compression cannot be specified for text columns. You can specify MAPPING VALUES compression for sorted and hashed indexes.

The MAPPING VALUES clause requires that you specify a range of values that can be stored in the column. The MAPPING VALUES clause therefore acts as a constraint, because it disallows values outside the specified range.

To create a MAPPING VALUES compressed index for columns that use TINYINT, SMALLINT, and INTEGER data types, use the MAPPING VALUES clause of the SQL CREATE INDEX statement for the column or columns being indexed (shown in Example 3–36). You can use the UNIQUE clause with an integer compressed index. In Example 3–36, PRODUCT_ID, YEAR_NUMBER, and PRODUCT_DESCR are the three columns that are defined with the UNIQUE clause.

Example 3–36 Setting MAPPING VALUES Index Compression for a SMALLINT Data Type Column

```
SQL> CREATE UNIQUE INDEX PS_DATE_2 ON PRODUCT_SCHEDULE
cont> (PRODUCT_ID,
cont>  YEAR_NUMBER  MAPPING VALUES 1970 to 2070,
cont>  PRODUCT_DESCR SIZE IS 20);
```

All index keys have an additional single bit added by Oracle Rdb. For most index key values, this bit is contained in a leading extra byte of which the low bit is the only significant bit. For numeric index keys for which MAPPING VALUES and ENABLE COMPRESSION are specified, this additional bit is a leading bit and no additional byte is required. This extra bit or byte is clear if there is an actual data value for the index key and set if the data value is missing or null.

The null bit is used to ensure that null index keys are sorted after non-null index keys and to provide data for the hash algorithm if hashed index keys are null.

Oracle Rdb automatically uses null bit compression for all index keys.

3.9.3.4 Run-Length Compression

When you specify run-length compression, Oracle Rdb compresses a sequence of space characters (octets) from text data types and binary zeros from nontext data types. (Different character sets have different representations of the space character. Oracle Rdb compresses the representation of the space character for the character sets of the columns comprising the index values.) Run-length compression is most useful when you have many sequences of space characters or binary zeros. You specify run-length compression by specifying the ENABLE COMPRESSION MINIMUM RUN LENGTH clause of the SQL CREATE INDEX statement. You can specify run-length compression for hashed and sorted indexes. Run-length compression can also be specified for system indexes, as described in Section 3.9.4.

To create a run-length compressed index for columns that use the CHAR or VARCHAR data types, use the ENABLE COMPRESSION MINIMUM RUN LENGTH clause of the SQL CREATE INDEX statement, as shown in Example 3–37. The value you specify in the MINIMUM RUN LENGTH clause indicates the minimum length of the sequence that Oracle Rdb should compress. For example, if you specify MINIMUM RUN LENGTH 2, Oracle Rdb compresses each sequence of two or more space characters or two or more binary zeros.

Example 3–37 Setting MINIMUM RUN LENGTH Index Compression for CHAR Data Type Columns

```
SQL> CREATE TABLE TELEPHONE_LIST
cont>  (NAME CHAR (60),
cont>   ADDRESS CHAR (141),
cont>   TELEPHONE_NUMBER CHAR (12),
cont>   TIME_DATE_OF_CALL DATE);
SQL>
SQL> CREATE INDEX TELEPHONE_CUSTOMER ON TELEPHONE_LIST
cont>  (NAME, ADDRESS, TELEPHONE_NUMBER)
cont>  ENABLE COMPRESSION
cont>  (MINIMUM RUN LENGTH 2);
```

When Oracle Rdb compresses a sequence of space characters or binary zeros, the sequence is replaced by the number of space characters or binary zeros specified by the mininum run-length value plus an extra byte that contains information about the number of space characters or binary zeros compressed for that sequence.

The storage savings from run-length compression can be significant. Consider the TELEPHONE_CUSTOMER index created in Example 3–37. If the TELEPHONE_CUSTOMER index were defined without specifying run-length compression, each uncompressed index key would be 216 bytes in length. Each 216-byte uncompressed index key consists of:

- 60 bytes for the NAME column (the first segment in the index)
- 141 bytes for the ADDRESS column (the second segment in the index)
- 12 bytes for the TELEPHONE_NUMBER column (the third segment in the index)
- 3 other bytes (each of the three index key segments has an extra byte that contains the null bit for that index key segment)

If the average name to be stored in the NAME column is only 13 characters in length, run-length compression will reduce the amount of space used to store the index keys.

Example 3–38 shows how Oracle Rdb compresses space characters for the index entry “Terry L Smith” when MINIMUM RUN LENGTH 2 is specified for the TELEPHONE_CUSTOMER index. In Example 3–38, the number sign (#) character represents a space character, and the c character represents a byte that contains compression information. The uncompressed index key segment is 60 characters in length, which includes the 13 characters for the name and 47 trailing space characters. When MINIMUM RUN LENGTH 2 is specified, Oracle Rdb replaces the 47 trailing space characters with 2 space characters (the value specified by the MINIMUM RUN LENGTH clause) and adds the extra byte with the compression information. The total size of the compressed index key segment is therefore only 16 bytes, which is a savings of 44 bytes compared to the uncompressed index key segment.

Example 3–38 Compression of an Index Key for the NAME Column of the TELEPHONE_CUSTOMER Index

```
Uncompressed index key entry:

Terry#L#Smith###############################################

Compressed index key entry with MINIMUM RUN LENGTH 2:
```

(continued on next page)

Example 3–38 (Cont.) Compression of an Index Key for the NAME Column of the TELEPHONE_CUSTOMER Index

```
Terry#L#Smith##c
^           ^  ^                                           ^
|           |  |                                           |
|           |  |                                           |
1          13 16                                          60
Number of characters in the index key
```

Similarly, if the average address to be stored in the ADDRESS column is only 18 characters in length, run-length compression will reduce the amount of space used to store the index keys. With an uncompressed index key segment for the ADDRESS column of 18 characters in length, there will be 123 trailing space characters (based on a column size of 141 characters for the ADDRESS column).

When MINIMUM RUN LENGTH 2 is specified, Oracle Rdb replaces the 123 trailing space characters with 2 space characters (the value specified by the MINIMUM RUN LENGTH clause) plus adds the extra byte with the compression information. The total size of the compressed index key segment is therefore only 21 bytes (18 bytes for the index key segment plus the 3 bytes resulting from the MINIMUM RUN LENGTH 2 clause), which is a savings of 120 bytes compared to the uncompressed index key segment.

The average size for each compressed index key in the TELEPHONE_CUSTOMER index would therefore be 52 bytes, consisting of:

- 16 bytes for the NAME column (the first segment in the index)
- 21 bytes for the ADDRESS column (the second segment in the index)
- 12 bytes for the TELEPHONE_NUMBER column (the third segment in the index)
- 3 other bytes (each of the three index key segments has an extra byte that contains the null bit for that index key segment)

Assume that you plan to store 100,000 entries in the TELEPHONE_CUSTOMER index. If you were to store the index entries in their uncompressed form, the storage space for the index entries would be 42188 disk blocks:

```
 100,000 index entries * 216 bytes per entry = 21,600,000 bytes

 21,600,000 bytes / 512 bytes per block = 42187.5 blocks
```

However, if you specified MINIMUM RUN LENGTH 2 when you created the index, the storage space for the 100,000 index key entries would be approximately 10157 blocks (some index key entries might be larger or smaller than the average 52-byte entry):

```
100,000 index entries * 52 bytes per entry = 5,200,000 bytes
5,200,000 bytes / 512 bytes per block = 10156.3 blocks
```

The storage savings from run-length compression in this case is approximately 32,030 blocks (42188 blocks minus 10157 blocks). Table 3–10 shows the storage space saved by enabling run-length compression for the TELEPHONE_CUSTOMER index when the index has 100,000 or 10 million index key entries.

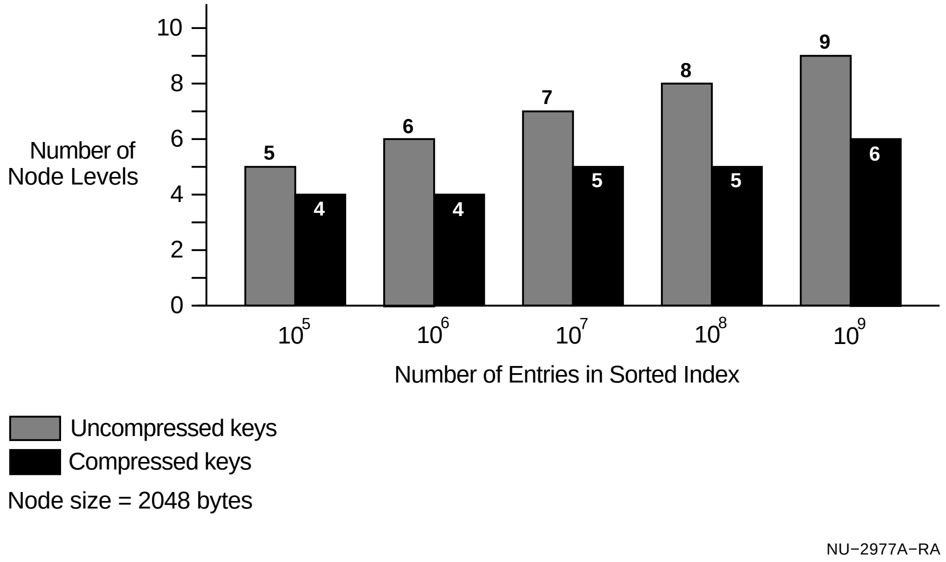
Table 3–10 Storage Savings for the TELEPHONE_CUSTOMER Index with Run-Length Compression Enabled

Number of Index Entries	Uncompressed Index Entries	Compressed Index Entries	Savings
100,000	42,188 blocks	10,157 blocks	32,030 blocks
10,000,000	4,218,750 blocks	1,015,625 blocks	3,203,125 blocks

Table 3–10 shows that an index with uncompressed keys takes up more space than the same index with compressed keys. Another benefit of compressing index keys is that more index keys can be stored in each sorted index node.

Figure 3–5 shows a sorted index created with a node size of 2048 bytes. When 100,000 uncompressed index key entries are stored, there will be 5 levels of nodes in the index. If the same 100,000 index keys are compressed, however, there will be only 4 levels of nodes in the index. Reducing the number of index node levels is beneficial from an I/O perspective because as the number of node levels is reduced, the number of I/O operations required to access index key entries is also reduced.

Figure 3–5 Reducing the Number of Index Node Levels by Specifying Run-Length Compression



One effect of storing more index keys in each node can be reduced concurrency. When more index keys fit in a node, you lock more index values each time you update a row. If you find that reducing the size of your index keys reduces concurrency, you can increase concurrency by reducing the node size so that each node contains fewer index keys.

You should have a good understanding of the data that will be stored in an index before you specify a MINIMUM RUN LENGTH value for the index. In some cases, you can inadvertently cause a compressed index to be larger than an uncompressed index by specifying a less than optimal MINIMUM RUN LENGTH value. In the following example, MINIMUM RUN LENGTH 1 has been specified for an index. The number sign (#) character represents a binary zero and the c character represents a byte that contains compression information.

```
Uncompressed index key entries:

1#2#3#4#5#6
2#3#4#5#6#7
3#4#5#6#7#8

Compressed index key entries with MINIMUM RUN LENGTH 1:
```

```
1#c2#c3#c4#c5#c6
2#c3#c4#c5#c6#c7
3#c4#c5#c6#c7#c8
^         ^    ^
|         |    |
|         |    |
1        11   16
Number of characters in the index key
```

Note that for this particular index, the compression that occurs when MINIMUM RUN LENGTH 1 is specified causes the compressed index keys to be larger than the uncompressed index keys (16 bytes for the compressed keys compared to 11 bytes for the uncompressed keys). The compressed index keys are larger because MINIMUM RUN LENGTH 1 tells Oracle Rdb to compress each binary zero with the number of zeros specified by the minimum run-length value. The result in this case is that Oracle Rdb replaces each single binary zero with one binary zero plus the extra byte that contains information on the number of binary zeros compressed (the single binary zero is replaced by two characters). Note that although the uncompressed index key entries take 11 bytes of storage space, Oracle Rdb can still store the 16-byte uncompressed index key entries. Oracle Rdb can store any compressed or uncompressed index key entry that is less than 255 bytes in size. If the size of a compressed index key exceeds 255 bytes at run time, you receive the following error message:

```
%RDMS-F-IKEYOVFLW, compressed IKEY for index index-name exceeds 255 bytes
```

If you receive this message, it means that the index key cannot be stored in the index; delete the index and define it again. To avoid receiving the same error message with the new index, you should specify that run-length compression is disabled for the new index, or you can enable run-length compression and specify a larger minimum run-length compression value.

Note that when you specify run-length compression, you cannot specify which characters are compressed, only the minimum length of the sequences you want Oracle Rdb to compress. Oracle Rdb determines which characters are compressed.

The SQL SHOW INDEXES statement shows the compression characteristics for defined indexes. The RMU Analyze Indexes command using the default Option=Normal qualifier shows how much space each index node uses for sorted indexes; see Section 3.9.5.1. You can see the results of index compression if you note the values displayed for individual nodes before and after index compression. Because sorted indexes are preallocated structures, this information is visible. Compression information is not readily visible for the dynamic hashed index structures because the Used/Avail values in the RMU Analyze Indexes Option=Normal display are the same. You must use the Option=Debug qualifier on the command line and inspect individual values for the hashed index structures before and after index compression to notice differences.

For more information on compressed indexes, see the *Oracle Rdb7 Guide to Database Design and Definition* and the *Oracle Rdb7 SQL Reference Manual*.

3.9.4 System Index Compression

When you create a database, you can specify that Oracle Rdb compress the system indexes. To do so, use the SYSTEM INDEX COMPRESSION IS ENABLED clause of the CREATE DATABASE statement. To enable system index compression for an existing database, you must export and import the database and specify the SYSTEM INDEX COMPRESSION IS ENABLED clause with the IMPORT statement.

For system indexes, Oracle Rdb uses run-length compression, which compresses a sequence of space characters from text data types and binary zeros (also called ASCII NUL) from nontext data types. It compresses any sequences of two or more spaces for text data types or two or more binary zeros for nontext data types. See Section 3.9.3.4 for more information on run-length compression.

Compressing system indexes results in reduced storage and improved I/O. Unless your applications often perform data definition concurrently, you should use compressed system indexes.

3.9.5 Gathering Index Information

This section describes the RMU Analyze and RMU Show commands that you can use to gather sorted and hashed index information. For general information on how to use the RMU Analyze command, refer to Section 2.1. Refer to Section 8.1.3.2 for more advice on analyzing indexes.

3.9.5.1 RMU Analyze Indexes Display

This section describes the format and content of the output when you use the RMU Analyze command and specify the Indexes and the Option [= Normal, Full, or Debug] qualifiers.

Using the RMU Analyze Indexes Option=Normal Command

When you use the RMU Analyze Indexes command with the Option=Normal qualifier and specify the EMPLOYEES_HASH hashed index, Oracle RMU displays the information shown in Example 3–39.

Example 3–39 RMU Analyze Indexes Option=Normal Command for a Hashed Index

```
$ RMU/ANALYZE/INDEXES mf_personnel EMPLOYEES_HASH /OPTION=NORMAL
----------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

----------------------------------------------------------------------------
 Hashed index EMPLOYEES_HASH for relation EMPLOYEES duplicates not allowed
 Max Level: 1,❶ Nodes: 69,❷ Used/Avail: 2797/2797 (100%),❸ Keys: 100,❹ Records: 100❺

----------------------------------------------------------------------------
```

The following list describes the RMU Analyze Indexes Option=Normal display. The callouts are keyed to fields in Example 3–39.

❶ Max Level:

The maximum number of levels in the index (1).

❷ Nodes:

The total number of nodes in the index (69). Because the display is of a hashed index, the nodes are hash buckets.

❸ Used/Avail:

The number of bytes used by the index (2797), the number of bytes available (2797), and the percent of the space used by the index (100%).

For hashed indexes, the Used/Avail value includes the index header and trailer information. Because the hashed index is a dynamic structure, its Used/Avail values are always the same and always show 100 percent usage.

For sorted indexes, the Used/Avail values do not include the index header and trailer information, which use 32 bytes.

❹ Keys:

The total number of unique keys in the index (100). Because in this example duplicates are not allowed, there are 100 unique keys for 100 data rows.

❺ Records:

The total number of index records with unique keys in the index (100).

When you use the RMU Analyze Indexes command with the Option=Normal qualifier and the DEPARTMENTS_INDEX sorted index, Oracle RMU displays the information shown in Example 3–40.

Example 3–40 RMU Analyze Indexes Option=Normal Command for a Sorted Index

```
$ RMU/ANALYZE/INDEXES mf_personnel DEPARTMENTS_INDEX /OPTION=NORMAL
------------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

------------------------------------------------------------------------------
 Index DEPARTMENTS_INDEX for relation DEPARTMENTS duplicates not allowed
 Max Level: 1, Nodes: 1, Used/Avail: 188/398 (47%), Keys: 26, Records: 26

------------------------------------------------------------------------------
```

In Example 3–40 for the DEPARTMENTS_INDEX sorted index, the maximum level is 1, there is one node (the root node), the index uses 188 of 398 possible bytes, and the nodes are 47 percent full. Because duplicates are not allowed, there are 26 unique keys for 26 data rows. Note that for sorted indexes, the amount of allocated space defaults to 430 bytes but usually not that much space is actually used for storing keys. For this reason, the percent used space is usually less than 100 percent.

For sorted indexes, the Used/Avail values do not include the index header and trailer information that accounts for 32 bytes. Therefore, the actual preallocated node size for this sorted index is 430 bytes, not 398 bytes. Because sorted indexes, unlike hashed indexes, are preallocated structures, this value (47 percent) is an actual percent usage value.

When you use the RMU Analyze Indexes command with the Option=Normal qualifier and a ranked sorted index that allows duplicates, Oracle RMU displays the information shown in Example 3–41.

Example 3–41 RMU Analyze Indexes Option=Normal Command for a Ranked Sorted Index

```
$ RMU/ANALYZE/INDEXES mf_personnel DEGREES_YEAR_RANKED /OPTION=NORMAL
------------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

------------------------------------------------------------------------------
 Index DEGREES_YEAR_RANKED for relation DEGREES duplicates allowed
 Max Level: 2, Nodes: 5, Used/Avail: 772/1990 (39%), Keys: 25, Records: 3
     Duplicate nodes: 0, Used/Avail: 0/0 (0%), Keys: 18, Maps: 18, Records: 162

------------------------------------------------------------------------------
```

When an index allows duplicates, the RMU Analyze Index command displays the following additional information:

- Duplicate nodes

For ranked sorted indexes, the number of overflow nodes. The number can be zero (0) even if the index contains duplicates. For non-ranked sorted indexes, the number of duplicate nodes. For hashed indexes, the number of duplicate nodes.

- Used/Avail

  The number of bytes used by duplicate nodes and number of bytes available in the duplicate nodes (the percentage of space used within the duplicate nodes of the index). This value can be zero (0) for a ranked sorted index if the number of duplicate nodes is zero.

- Keys

  The total number of duplicate keys in the index.

- Maps

  For ranked sorted indexes only, the number of bit maps used to represent the dbkeys that point to duplicate index key data. This field does not appear for non-ranked sorted indexes or hashed indexes.

- Records

  The total number of duplicate records in the index.

When you use the RMU Analyze Indexes command with the qualifier Option=Normal and specify the JOB_HISTORY_HASH hashed index, which allows duplicates, Oracle RMU displays the information shown in Example 3–42.

Example 3–42 RMU Analyze Indexes Option=Normal Command for a Hashed Index (Duplicates Allowed)

```
$ RMU/ANALYZE/INDEXES mf_personnel JOB_HISTORY_HASH /OPTION=NORMAL
------------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

------------------------------------------------------------------------------
 Hashed index JOB_HISTORY_HASH for relation JOB_HISTORY duplicates allowed
 Max Level: 1, Nodes: 69, Used/Avail: 2797/2797 (100%), Keys: 100, Records: 20
     Duplicate nodes: 80, Used/Avail: 2992/7360 (41%), Keys: 80, Records: 254

------------------------------------------------------------------------------
```

In Example 3–42 for the JOB_HISTORY_HASH hashed index, the maximum level is 1, there are 69 nodes (hash buckets) using 2797 bytes or 100 percent of available space. There are a total of 100 keys in the index. There are 20 records with unique keys. There are 254 records with keys that are not unique, and 80 of the 100 keys are not unique. The average number of duplicates per non-unique key for the index is calculated as 2.175 or (254/80) –1 and the average number of duplicates for the index as a whole is calculated as 1.74 or ((254+20)/100) –1. In addition, the 80 duplicate node records are 41 percent filled with 2992 bytes of a possible 7360 bytes.

For sorted indexes, the value for the number of levels indicates how many levels deep the B-tree index has become.

For hashed indexes, the value for the number of levels is the length of the longest hash bucket overflow chain for the index. Overflows occur when there is insufficient space on the page to make additional entries in the hash bucket, and an overflow hash bucket is then created on an adjacent page, where there is sufficient space. Overflows may be more prevalent when entire hash structures for both parent and child records reside on the same page as their data rows that use a PLACEMENT VIA INDEX clause. The value for the number of index levels may or may not indicate how many I/O operations are required to find a particular value when the system uses the index for retrieval. However, the results of an RMU Analyze Placement command using the Option=Full or Option=Debug qualifier and an inspection of the Minimum I/O Path Length by Frequency histogram or the minimum I/O path length values, will provide an accurate estimate of whether or not all the desired information resides in the buffer.

If Oracle Rdb reads the top and bottom level of the sorted index and the data page, three I/O operations would be necessary to make the retrieval. However, the top level of the index normally stays in the buffer. If there are enough buffers, the lower levels remain in the buffer as well. In this case, only one I/O operation would be required to retrieve the row.

For hashed indexes that are stored in the same storage area and on the same page as the data using the PLACEMENT VIA INDEX clause of the SQL CREATE and ALTER STORAGE MAP statements, a minimum of one I/O operation is required to retrieve the data row. When duplicates are allowed, duplicate node records are created. If there are many duplicate records and the page size is too small, data rows may be placed on the page to which they hash, but portions of the hash structure (hash buckets) will overflow to nearby pages because there is not sufficient space to make additional entries in the hash bucket. Over time, as the database pages fill up, more overflows may occur. As a result, performance may drop as more I/O operations become necessary to gather the requested information.

The display from the RMU Analyze Indexes Option=Normal command, as shown in Example 3–43, provides an extra line of detail for compressed indexes. The last line of the display appears only when the index is a compressed index.

Example 3–43 RMU Analyze Indexes Option=Normal Command for a Compressed Index

```
$ RMU/ANALYZE/INDEXES test_db COMPRESSED_IND /OPTION=NORMAL
0-------------------------------------------------------------------------------
 Index COMPRESSED_IND for relation NEW_TABLE duplicates allowed
 Max Level: 1, Nodes: 1, Used/Avail: 155/398 (39%), Keys: 8, Records: 8
     Duplicate nodes: 0, Used/Avail: 0/0 (0%), Keys: 0, Records: 0
     Total Comp/Uncomp IKEY Size: 179/256, Compression Ratio:  .70
0
0-------------------------------------------------------------------------------
```

In the last line of Example 3–43, the 179 indicates the total byte count of compressed leaf (Level 1 nodes only) index keys. The 256 indicates the total byte count that would be consumed if the index were not compressed. A compression factor greater than 1.0 indicates that the compressed index keys occupy more space than the uncompressed index keys. Section 3.9.3.4 describes how a compressed index key can occupy more space than an uncompressed index key if run-length compression is used.

Using the RMU Analyze Indexes Option=Full Command

When you use the RMU Analyze Indexes command and specify the Option=Full qualifier, information for all levels of the index is displayed. For the DEPARTMENTS_INDEX sorted index, the index is only one level, as shown in Example 3–44.

Example 3–44 Using the RMU Analyze Indexes Option=Full Command on a Sorted Index

```
$ RMU/ANALYZE/INDEXES mf_personnel DEPARTMENTS_INDEX /OPTION=FULL
-------------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

-------------------------------------------------------------------------------
 Index DEPARTMENTS_INDEX for relation DEPARTMENTS duplicates not allowed
 Max Level: 1, Nodes: 1, Used/Avail: 188/398 (47%), Keys: 26, Records: 26

-------------------------------------------------------------------------------
```

If the index has several levels, the display shows information for each level. Because this index has only one level, the information is repeated.

Using the RMU Analyze Indexes Option=Debug Command

When you use the RMU Analyze Indexes command and specify the Option=Debug qualifier, detailed information for each index node and index record is displayed. Example 3–45 shows the output for the EMPLOYEES_HASH hashed index.

Example 3–45 RMU Analyze Indexes Option=Debug Command for a Hashed Index

```
$ RMU/ANALYZE/INDEXES mf_personnel EMPLOYEES_HASH /OPTION=DEBUG
0------------------------------------------------------------------------------
0
0 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
0
0   ID   Hash Dup Name                           Relation
0------------------------------------------------------------------------------
4      62   T   F  EMPLOYEES_HASH                 EMPLOYEES
4      60   T   F  EMPLOYEES_HASH                 EMPLOYEES
4      58   T   F  EMPLOYEES_HASH                 EMPLOYEES
0
0------------------------------------------------------------------------------
0
0   ID        DB KEY       LEVEL    SIZE    KEY-LEN    KEY
0------------------------------------------------------------------------------
0
5  58 0058:0000000002:002     1 0051/0051
5  58 0063:0000000002:001     0    0/   0     (006)"003030313635"
5  58 0063:0000000002:003     0    0/   0     (006)"003030313930"
5  58 0058:0000000005:002     1 0032/0032
5  58 0063:0000000005:001     0    0/   0     (006)"003030313837"
5  58 0058:0000000007:002     1 0070/0070
5  58 0063:0000000007:001     0    0/   0     (006)"003030313639"
5  58 0063:0000000007:003     0    0/   0     (006)"003030313736"
5  58 0063:0000000007:004     0    0/   0     (006)"003030313938"
5  58 0058:0000000011:002     1 0032/0032
   .
   .
   .
5  60 0060:0000000003:002     1 0032/0032
5  60 0064:0000000003:001     0    0/   0     (006)"003030323133"
5  60 0060:0000000004:002     1 0032/0032
5  60 0064:0000000004:001     0    0/   0     (006)"003030323139"
5  60 0060:0000000005:002     1 0051/0051
5  60 0064:0000000005:001     0    0/   0     (006)"003030323235"
5  60 0064:0000000005:003     0    0/   0     (006)"003030323430"
5  60 0060:0000000006:002     1 0032/0032
   .
   .
   .
```

(continued on next page)

Example 3–45 (Cont.) RMU Analyze Indexes Option=Debug Command for a Hashed Index

```
5  62 0062:0000000008:002     1 0032/0032
5  62 0065:0000000008:001     0    0/   0     (006)"003030343135"
5  62 0062:0000000018:002     1 0032/0032
5  62 0065:0000000018:001     0    0/   0     (006)"003030343335"
5  62 0062:0000000021:002     1 0032/0032
5  62 0065:0000000021:001     0    0/   0     (006)"003030343035"
5  62 0062:0000000026:002     1 0032/0032
5  62 0065:0000000026:001     0    0/   0     (006)"003030343138"
5  62 0062:0000000032:002     1 0032/0032
5  62 0065:0000000032:001     0    0/   0     (006)"003030343731"
5  62 0062:0000000046:002     1 0032/0032
5  62 0065:0000000046:001     0    0/   0     (006)"003030343136"
0-------------------------------------------------------------------------------
0
0 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
0
0-------------------------------------------------------------------------------
0 Hashed index EMPLOYEES_HASH for relation EMPLOYEES duplicates not allowed
0 Max Level: 1, Nodes: 69, Used/Avail: 2797/2797 (100%), Keys: 100, Records: 100
0
0     Level: 1, Nodes: 69, Used/Avail: 2797/2797 (100%), Keys: 100, Records: 100
0
0-------------------------------------------------------------------------------
```

The RMU Analyze Indexes command with the Option=Debug qualifier displays detailed index node and index record information. The output header is divided into two parts, general and detailed information. The headers are followed by detailed index record information. The last part of the output displays information identical to the RMU Analyze Indexes command when you specify the Option=Full qualifier. Note that if an index is stored in more than one logical area, information for all logical areas is displayed in the output.

The following list describes the RMU Analyze Indexes Option=Debug display. Entries in parentheses that follow each field description are keyed to Example 3–45, which uses the EMPLOYEES_HASH index. The description references logical area 58 and the first line of detailed index information for this logical area.

- General index information
  - ID

    The logical area ID for the index (58).
  - Hash

A coded field: T = TRUE for hashed index, F = FALSE for sorted index (T).

  - Dup

    A coded field: T = TRUE for duplicates allowed, F = FALSE for duplicates not allowed (F).

  - Name

    The name of the index (EMPLOYEES_HASH).

  - Relation

    The name of the table on which the index is defined (EMPLOYEES).

- Detailed index information

  - ID

    The logical area ID for the index (58).

  - DB KEY

    The dbkey for the record (0058:0000000002:002); it is composed of three parts: the logical area ID (0058), the page number (0000000002), and the line on the page where the record is stored (002).

  - LEVEL

    The level of the index record (1); this indicates a node record or, in this case, a hash bucket.

  - SIZE

    For hashed indexes, the total size (in bytes) of each hash bucket and following the slash (/), the amount of space (in bytes) allocated for the hash bucket (0051 bytes/0051 bytes).

    For sorted indexes (see Example 3–46), the total amount of space used by all index records and following the slash (/), the amount of space allocated for the node record.

  - KEY-LEN

    The length of the key, in bytes (6).

    For sorted indexes (see Example 3–46), the values in parentheses have the following meaning. When the key value is compared to the preceding key value, the first set of three digits (the prefix) indicates the number of bytes in the preceding key that are the same. The second set of three digits (the suffix) following the plus (+) sign indicates the number of bytes that are different from the preceding key.

For example, for the key-len of (000+005), the value 000 indicates no bytes are the same and 5 bytes are different from the preceding key because there is no preceding key. In the next line, the key-len is (001+004). The value 001 indicates that the first byte (00) is the same in the preceding key, while the remaining 4 bytes are different. For the next line, the key-len is (003+002). The value 003 indicates that the first 3 bytes (00454C) are the same as the preceding key, while the remaining 2 bytes are different.

This information is useful for key compression where the prefix value is compressed because it repeats. In traversing a B-tree index, the prefix values are picked up at each branch of the tree until the leaf node is reached, which is where the unique suffix value is found. The total key-len is 5 bytes for the DEPARTMENTS_INDEX sorted index.

- KEY

  The actual key printed as a HEX string (003030313635).

In Example 3–45 for the EMPLOYEES_HASH hashed index, note that the index is partitioned across three logical areas (58, 60, and 62) for the table EMPLOYEES. Dbkeys are listed by logical area and by page number within logical area. The key is 5 bytes in size, plus 1 byte for the null bit vector is equal to 6 bytes, plus 1 byte for the key size, or a total of 7 bytes. The total size of each hash bucket is displayed; for example, 0051 for the first hash bucket. The 51 bytes for this hash bucket include 13 bytes of fixed header information for the hash bucket. The hash bucket also contains 19 bytes of information for each data row associated with this hash bucket. These 19 bytes include:

- 7 bytes for the key: length + null bit vector + key size of the row
- 4 bytes for the duplicate count associated with the row
- 8 bytes for the dbkey pointer to the row (or to the duplicate node if the index allows duplicates)

This means the hash bucket increases in increments of 19 bytes. The smallest hash bucket possible for this index is 32 bytes (19 bytes for one data row + 13 bytes of fixed header information for the hash bucket). This hash bucket has two data rows associated with it, so the size of the hash bucket is 51 bytes. The actual dbkey and key value is displayed for each data row.

When you use the RMU Analyze Indexes command and specify the Option=Debug qualifier and the DEPARTMENTS_INDEX sorted index, the information displayed is shown in Example 3–46.

Example 3–46 RMU Analyze Indexes Option=Debug Command for a Sorted Index

```
$ RMU/ANALYZE/INDEXES mf_personnel DEPARTMENTS_INDEX /OPTION=DEBUG
0-------------------------------------------------------------------------------
0
0 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
0
0   ID   Hash Dup Name                          Relation
0-------------------------------------------------------------------------------
4      72   F   F  DEPARTMENTS_INDEX              DEPARTMENTS
0
0-------------------------------------------------------------------------------
0
0   ID        DB KEY       LEVEL    SIZE    KEY-LEN    KEY
0-------------------------------------------------------------------------------
0
5  72 0072:0000000002:002     1 0188/0430     (000)""
5  72 0073:0000000002:001     0    0/   0 (000+005)"0041444D4E"
5  72 0073:0000000002:003     0    0/   0 (001+004)"00454C454C"
5  72 0073:0000000002:004     0    0/   0 (003+002)"00454C4753"
5  72 0073:0000000002:005     0    0/   0 (003+002)"00454C4D43"
5  72 0073:0000000002:006     0    0/   0 (002+003)"00454E4720"
5  72 0073:0000000002:007     0    0/   0 (001+004)"004D424D46"
5  72 0073:0000000002:008     0    0/   0 (004+001)"004D424D4E"
5  72 0073:0000000002:009     0    0/   0 (004+001)"004D424D53"
5  72 0073:0000000003:001     0    0/   0 (002+003)"004D43424D"
5  72 0073:0000000003:002     0    0/   0 (004+001)"004D434253"
5  72 0073:0000000003:003     0    0/   0 (002+003)"004D475654"
5  72 0073:0000000003:004     0    0/   0 (002+003)"004D4B5447"
5  72 0073:0000000003:005     0    0/   0 (002+003)"004D4E4647"
5  72 0073:0000000003:006     0    0/   0 (002+003)"004D534349"
5  72 0073:0000000003:007     0    0/   0 (003+002)"004D534D47"
5  72 0073:0000000003:008     0    0/   0 (002+003)"004D54454C"
5  72 0073:0000000003:009     0    0/   0 (001+004)"005045524C"
5  72 0073:0000000003:010     0    0/   0 (004+001)"0050455253"
5  72 0073:0000000003:011     0    0/   0 (002+003)"005048524E"
5  72 0073:0000000003:012     0    0/   0 (002+003)"0050524D47"
5  72 0073:0000000003:013     0    0/   0 (001+004)"0053414C45"
5  72 0073:0000000003:014     0    0/   0 (002+003)"0053455552"
5  72 0073:0000000003:015     0    0/   0 (002+003)"0053554E45"
5  72 0073:0000000003:016     0    0/   0 (003+002)"0053555341"
5  72 0073:0000000003:017     0    0/   0 (004+001)"005355534F"
5  72 0073:0000000002:010     0    0/   0 (003+002)"0053555745"
0-------------------------------------------------------------------------------
```

(continued on next page)

Example 3–46 (Cont.) RMU Analyze Indexes Option=Debug Command for a Sorted Index

```
0
0 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
0
0---------------------------------------------------------------------------
0 Index DEPARTMENTS_INDEX for relation DEPARTMENTS duplicates not allowed
0 Max Level: 1, Nodes: 1, Used/Avail: 188/398 (47%), Keys: 26, Records: 26
0
0     Level: 1, Nodes: 1, Used/Avail: 188/398 (47%), Keys: 26, Records: 26
0
0---------------------------------------------------------------------------
```

Example 3–46 shows the DEPARTMENTS_INDEX sorted index for the DEPARTMENTS table. The index is a non-ranked sorted index.

The entire index (26 records) is located on pages 2 and 3 in logical area 72 and uses 188 bytes of a possible 430 bytes or the node record is 47 percent full. Note that due to index compression, the node size has decreased in size from 214 bytes to 188 bytes and the percent fullness of the node records has dropped from 54 to 47 percent. Also note that the used/avail value in the summary information at the end of the output does not include the index header and trailer information, which accounts for 32 bytes. This value is shown for each node record in the detailed part of the output. The number of bytes used by the index is calculated as follows: the key is 4 bytes plus a null byte for a total of 5 bytes. The key length is 1 byte for the number of bytes, less the prefix length, that are stored in the key data; and the prefix length is 1 byte for the number of bytes of the previous entry that are prefixed to this one, for a total of 2 bytes. There are 26 data rows multiplied by 7 (5+1+1) or a total of 182 bytes. Add 32 bytes for index header and trailer information for the index node to the 182 bytes for a total of 214 bytes used. Index compression reduces the number of bytes used to 188 bytes used.

For detailed information on sizing ranked and non-ranked sorted indexes, see the *Oracle Rdb7 Guide to Database Design and Definition*.

For sorted indexes, you should pay close attention to the length of the index node and to the number of duplicates, if they are allowed. If the length of the index is large, you are using a lot of system resources to keep the index on the table. At the same time, if you allow duplicates and there are many duplicate values for the index, the efficiency of the index is reduced. You may want to do further testing to determine if the index improves performance enough to justify the overhead required to store and update it. You may also gain the advantage of using the index by defining multisegmented indexes to distribute access across more index nodes than are currently available.

Example 3–47 shows the display for the RMU Analyze Indexes Option=Debug command when a compressed index is specified.

Example 3–47 RMU Analyze Indexes Option=Debug Command for a Compressed Index

```
$ RMU/ANALYZE/INDEXES test_db COMPRESSED_IND /OPTION=DEBUG
0------------------------------------------------------------------------------
0
0 Indices for database  - $DUA3:[ORION]TEST_DB;
0
0   ID   Hash Dup Name                          Relation
0------------------------------------------------------------------------------
4      13   F   T  COMPRESSED_IND                  NEW_TABLE
0
0------------------------------------------------------------------------------
0
0   ID        DB KEY       LEVEL    SIZE    KEY-LEN    KEY
0------------------------------------------------------------------------------
0
5  13 0004:0000000146:005     0    0/   0  (006+025)"0052444224524C435F434F4E535
5241494E545F4E414D455F4E445820202020"
5  13 0004:0000000147:001     0    0/   0  (009+017)"0052444224524C435F4649454C4
5F4E414D455F4E4458202020202020202020"        ^          ^
        .                                    |          |
        .                                    |           Hexadecimal dumps
        .                                    |           (uncompressed keys)
                                              These lengths are for the
                                              compressed index key
```

The RMU Analyze Indexes Option=Debug command uncompresses index keys from compressed indexes before display. The hexadecimal output displayed shows the uncompressed index key. However, the length reported for each index key is the length of the compressed index key, as shown in Example 3–47.

3.9.5.2 Gathering Index Information Using the Performance Monitor

The Performance Monitor provides the following screens to help you gather information about sorted and hashed indexes:

- The Hash Index Statistics screen

  The Hash Index Statistics screen monitors the update and retrieval activity of a database's hashed indexes. It indicates the total number of key insertions and deletions. It also indicates the number of scans that were opened. For retrievals (successful fetches), the screen indicates the total number of nodes (either bucket fragments or duplicate nodes) that were fetched.

You can also refer to Section 8.1.3.4 and Section 8.1.3.5 for more advice on analyzing hashed indexes.

- The Index Statistics (Retrieval) screen

  The Index Statistics (Retrieval) screen monitors how much retrieval activity is taking place in a database's sorted indexes. Oracle Rdb often uses direct index lookups and index scans to access records in the database. The Index Statistics (Retrieval) screen monitors these operations as well as the number of index nodes fetched.

- The Index Statistics (Insertion) screen

  The Index Statistics (Insertion) screen monitors the update activity of a database's sorted indexes during insertions; that is, when you insert or modify an index key field or when you use the SQL CREATE INDEX statement on a table. This screen also indicates in which type of index node the insertions occur and displays node creations by node type. By examining this screen, you can monitor how a database balances its sorted indexes after insertions into the database.

- The Index Statistics (Removal) screen

  The Index Statistics (Removal) screen monitors the update activity of a database's sorted indexes when you perform any removal operation; that is, delete or modify an index key field or drop an index. This screen indicates from which type of index node the removals occur. It also shows node deletions by node type. This screen lets you monitor how a database balances its sorted indexes when nodes are removed from the indexes.

For information about each screen and each field in each screen, see the Peformance Monitor help.

3.9.6 Sorted Index Structure

The sorted index structure is based on a tree structure and contains nodes. Each sorted index (system or user) has a separate index structure or B-tree. Each B-tree structure is created by linking index nodes together in a balanced hierarchical structure. These nodes are also horizontally linked in low-to-high key value. The links between the nodes are created using dbkeys.

The type of nodes in the B-tree depends on whether the index is a ranked or non-ranked sorted index.

Ranked sorted indexes contain the following types of nodes:

- Index key nodes

Index key nodes contain unique index keys and pointers to other nodes. A B-tree structure can have many levels of index key nodes. Level 1 nodes, also known as **leaf nodes**, point to rows in the table or to overflow nodes. Level 2 (and above) nodes point to lower-level index nodes.

With ranked sorted indexes, Oracle Rdb compresses duplicates using byte-aligned bitmap compression. It takes a list of dbkeys that point to data rows with the same index values and stores the list as a compressed bitmap in the leaf node. If you store duplicate entries, Level 1 nodes contain the compressed bitmaps of dbkeys.

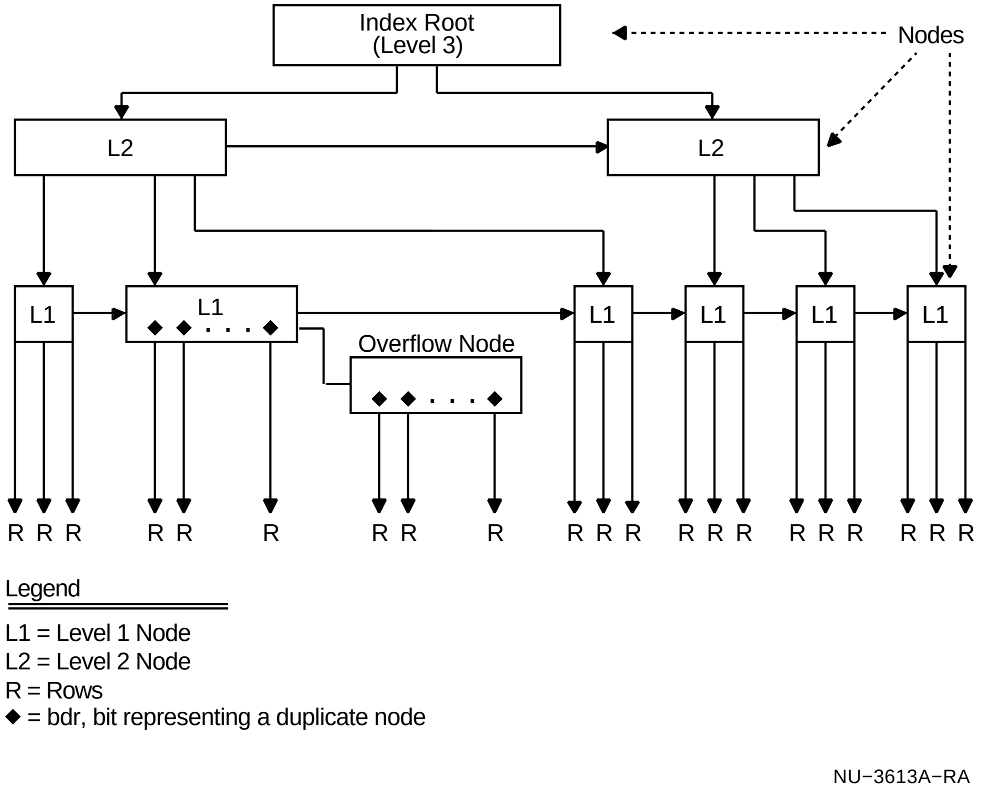
- Overflow nodes

  Oracle Rdb creates overflow nodes when the compressed bitmaps of duplicates in the leaf node overflow that node. The **overflow nodes** contain the continuation of the compressed bitmap stored in the leaf node and a pointer to the next overflow node.

Because index nodes are themselves rows, they are subject to row locking.

Figure 3–6 illustrates a B-tree index structure for a ranked sorted index.

Figure 3–6 B-Tree Index Structure for a Ranked Sorted Index



The level 2 (and above) nodes contain:

- A record type header that identifies index nodes and overflow nodes.
- Two fragmentation flags.
- A storage area ID that identifies the logical area of the index node.
- A level type that identifies the level of the B-tree structure in which the index node resides.
- Index key values with prefix and suffix compression.
- The entry cardinality of the row and leaf cardinalities.
- A list of dbkeys stored as a compressed bitmap. The dbkeys point to the data rows with the same index values.

- A pointer to the next node on the right at the same level and higher key value.
- Pointers to lower level nodes.

The level 1 nodes (the level closest to the data rows) contain:

- A record type header that identifies index nodes and overflow nodes.
- Two fragmentation flags.
- A storage area ID that identifies the logical area of the index node.
- A level type that identifies the level of the B-tree structure in which the index node resides.
- Index key value with prefix compression (leading bytes of the key value that repeat from one entry to the next are not stored).
- If the node contains duplicates, pointers to data rows that use compressed dbkeys storing the relative location of the row. If there are duplicates, the pointers are not used until the duplicates overflow. At that time, pointers indicate the location of the overflow dbkey.
- The entry cardinality of the row.
- If the node contains duplicates, a list of dbkeys stored as a compressed bitmap. The dbkeys point to the data rows with the same index values.
- Pointers to overflow nodes, if necessary. (Oracle Rdb creates overflow nodes only when the list of duplicates overflows the index node.)
- A pointer to the next level 1 node on the right (higher key value).

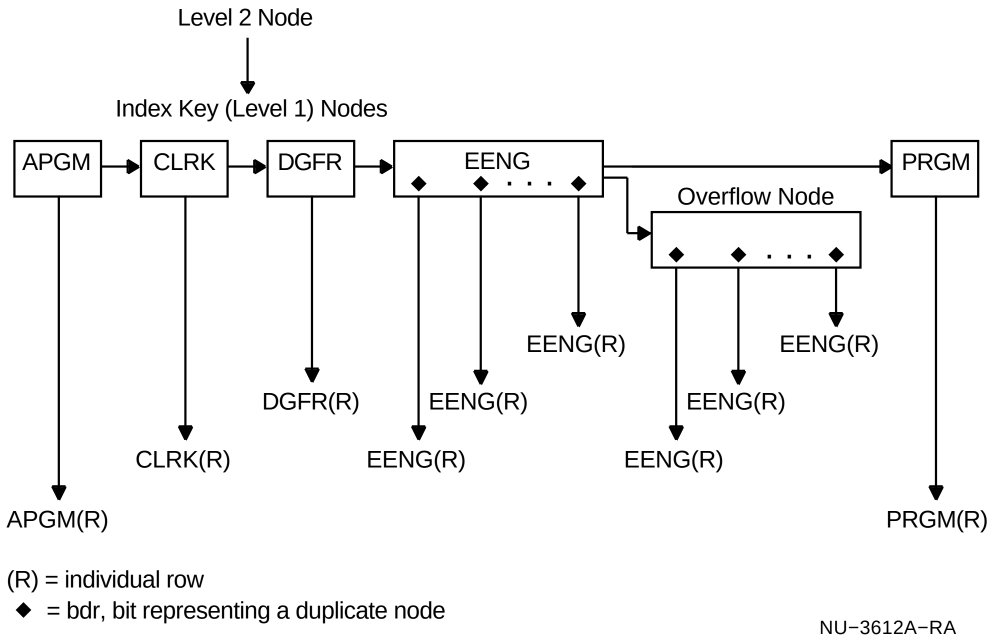
The overflow nodes contain:

- A record type header that identifies index nodes and duplicate nodes.
- Two fragmentation flags.
- A storage area ID that identifies the logical area of the index node.
- A list of dbkeys stored as a compressed bitmap. The dbkeys point to the data rows with the same index values.
- A pointer to the next overflow node in the chain.

For duplicate records, instead of pointing to individual records, the entry contains a list of dbkeys that, in turn, point to all the data rows with identical index key values. When many data rows have the same index key value, more than one overflow node might be required. This is called an **overflow chain**.

Figure 3–7 shows part of a B-tree structure with an overflow node.

Figure 3–7 Overflow Index Nodes



Non-ranked sorted indexes contain the following types of nodes:

- Index key nodes

  Index key nodes contain unique index keys and pointers to other nodes. A B-tree structure can have many levels of index key nodes. Level 1 nodes, also known as leaf nodes, point to rows in the table or to duplicate index rows. Level 2 (and above) nodes point to lower-level index nodes.

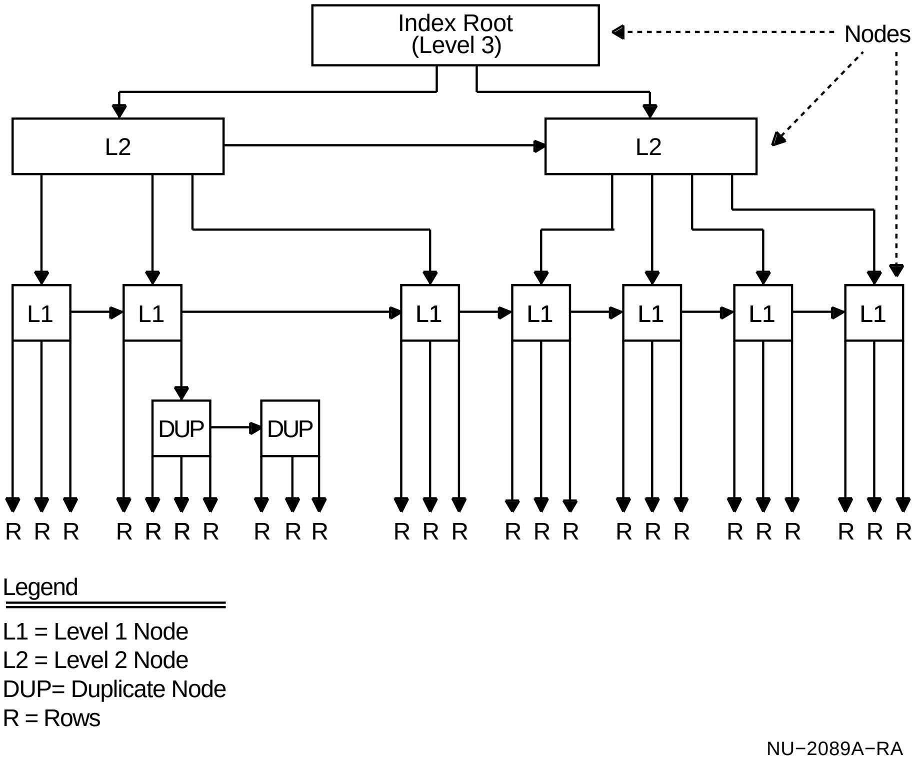
- Duplicate nodes

  Oracle Rdb creates duplicate index nodes when you define the index without using the UNIQUE keyword and you store a duplicate key value. The **duplicate nodes** contain a list of the duplicate row dbkeys.

Because index nodes are themselves rows, they are subject to row locking.

Figure 3–8 illustrates a B-tree index structure for a non-ranked sorted index.

Figure 3–8 B-Tree Index Structure for a Non-Ranked Sorted Index



The level 2 (and above) nodes contain:

- A record type header that identifies index nodes and duplicate nodes.
- Two fragmentation flags.
- A storage area ID that identifies the logical area of the index node.
- A level type that identifies the level of the B-tree structure in which the index node resides.
- Index key values with prefix and suffix compression.
- A pointer to the next node on the right at the same level and higher key value.
- Pointers to lower level nodes.

The level 1 nodes (the level closest to the data rows) contain:

- A record type header that identifies index nodes and duplicate nodes.
- Two fragmentation flags.
- A storage area ID that identifies the logical area of the index node.
- A level type that identifies the level of the B-tree structure in which the index node resides.
- Index key value with prefix compression (leading bytes of the key value that repeat from one entry to the next are not stored).
- Pointers to data rows that use compressed dbkeys storing the relative location of the row.
- Pointers to duplicate nodes that use dbkeys with negative logical area numbers.
- A pointer to the next level 1 node on the right (higher key value).

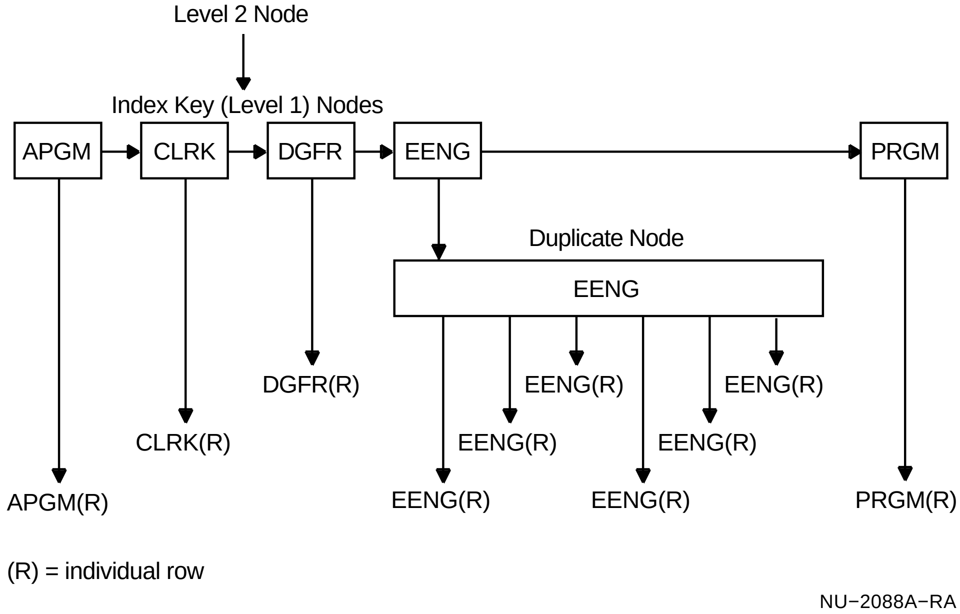
The duplicate nodes contain:

- A record type header that identifies index nodes and duplicate nodes.
- Two fragmentation flags.
- A storage area ID that identifies the logical area of the index node.
- Pointers to the data rows with same index key value.
- A pointer to the next duplicate node in the duplicate chain.

For duplicate node records, instead of pointing to individual records, the B-tree structure points to a duplicate node that, in turn, points to all the data rows with identical index key values. When many data rows have the same index key value, more than one duplicate node might be required. This is called a **duplicate chain**.

Figure 3–9 shows a duplicate node.

Figure 3–9 Duplicate Index Nodes

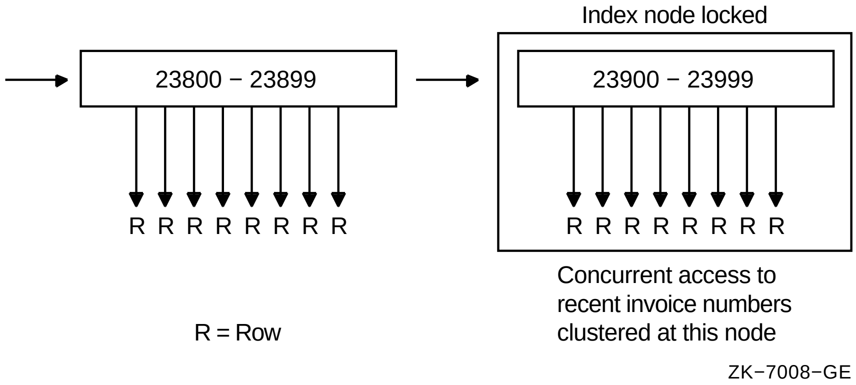


3.9.6.1 Reducing Locking of Chronological Keys

One common example of index design that increases lock contention in index structure is a **chronological key**; that is, an indexed key value stored in the database over time as its value increases. Access to key values, therefore, tends to be within the same few nodes of the index structure.

A good example of this type of key is an INVOICE_NUMBER column. Recent invoice numbers tend to contain larger values than older invoice numbers, and access to invoices may be clustered more toward recent additions. Because most access to these indexed columns is concentrated in a few nodes, as in Figure 3–10, updating the index while there are other users in the table can restrict database concurrency.

Figure 3–10 Clustered Access Contention at One Index Node



This lock contention problem has several solutions:

- You can concatenate another key value to the invoice number. This should be a column you would ordinarily need to satisfy the query. The goal is to have a value in the key that distributes the values of the multisegmented key more evenly throughout the entire index, instead of clustering access on one side. In the case of the INVOICE_NUMBER column, a multisegmented index that contains the CUSTOMER_NUMBER and INVOICE_NUMBER columns would be suitable. The SQL CREATE INDEX statement in Example 3–48 creates this multisegmented index. The trade-off with this approach is that new, redundant data must be stored in the index structure, and requires extra storage area space to maintain.

Example 3–48 Reducing Lock Contention When Using Chronological Keys

```
SQL> CREATE UNIQUE INDEX CUSTOMER_INVOICE ON INVOICES
cont>   (CUSTOMER_NUMBER,
cont>   INVOICE_NUMBER);
```

- You can preallocate the chronological keys at night in a batch run and then modify the rows with real data during online operation. This avoids the index contention, but may lead to fragmentation.
- You can consider the possibility of using a hashed index, which does not have problems with chronological keys.

3.9.6.2 Reducing Locking of Duplicate Nodes Key

Another lock contention problem results from a very small number of key values for the table, with a large number of duplicates, that are frequently updated. Because one index node can maintain many duplicate key values, updating just one key may lock the entire node.

A common example that illustrates this duplicates problem is a mailing list with an index defined for the 2-character STATE code and many duplicates for each index node. It is possible that the top node of the index will contain *all* the key values. When a user needs to update one of the duplicate values, the top node (the root of the tree) is locked for this update transaction and denies other users access to the table.

A solution to this problem is to concatenate the STATE code column with a POSTAL_CODE column. This multisegmented key distributes the key values across a larger, wider index structure and helps to reduce lock contention. The trade-off is that more storage space is needed to maintain the larger index structure.

The following SQL CREATE INDEX statement and RMU Analyze Indexes command output shows that an index has been defined for the STATE column in the EMPLOYEES table of the mf_personnel database. A second, multisegmented index has also been defined using two columns, STATE and POSTAL_CODE, to distribute the values across more nodes of the index, as shown in Example 3–49.

Example 3–49 Reducing Lock Contention When Many Duplicates Exist

```
SQL> CREATE INDEX STATE_IDX ON EMPLOYEES
cont> (STATE)
cont> TYPE IS SORTED;
SQL>
SQL> CREATE INDEX STATE_POSTAL_IDX ON EMPLOYEES
cont> (STATE,
cont> POSTAL_CODE)
cont> TYPE IS SORTED;
SQL> COMMIT;
SQL> EXIT
```

(continued on next page)

Example 3–49 (Cont.) Reducing Lock Contention When Many Duplicates Exist

```
$ RMU/ANALYZE/INDEXES mf_personnel STATE_IDX, STATE_POSTAL_IDX
-------------------------------------------------------------------------------
 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

-------------------------------------------------------------------------------
 Index STATE_IDX for relation EMPLOYEES duplicates allowed
 Max Level: 1, Nodes: 1, Used/Avail: 25/398 (6%), Keys: 3, Records: 1
     Duplicate nodes: 3, Used/Avail: 792/868 (91%), Keys: 2, Records: 99

-------------------------------------------------------------------------------
 Index STATE_POSTAL_IDX for relation EMPLOYEES duplicates allowed
 Max Level: 2, Nodes: 3, Used/Avail: 314/1194 (26%), Keys: 35, Records: 9
     Duplicate nodes: 24, Used/Avail: 728/1392 (52%), Keys: 24, Records: 91

-------------------------------------------------------------------------------
```

Although the index structure is designed to allow increased concurrent access to the EMPLOYEES table, access is clustered around three duplicate nodes, preventing access to a large section of the index. When you define an index for a column that contains duplicate values, you may discover that a table has many rows, but only a few values for a column for which you have defined an index. In this case, using STATE_POSTAL_IDX is beneficial because it allows other transactions to access the EMPLOYEES table to retrieve rows not locked by this transaction.

The trade-off in this method of indexing is that Oracle Rdb requires extra storage space for the distributed values for the multisegmented index keys. Remember that you can define indexes for one or more columns, use them for specific tasks, and delete them when they are no longer required. For example, you can use indexes for temporary reporting purposes. Create the indexes you need before you run a read-only report and then delete them before the database is used for production. In this way, you benefit from increased concurrent access while you maintain efficient use of disk storage.

3.9.6.3 Clustering Indexes

A cluster index is a sorted index that is used to place presorted table rows in a storage area so that rows are stored in approximately the same order in which they appear in the index. This can significantly improve performance for queries requiring range retrievals. For example, if you create an ascending index on the LAST_NAME column and then specify that the table will be stored using that sorted index with a PLACEMENT VIA INDEX clause in an SQL CREATE STORAGE MAP statement, Oracle Rdb attempts to store the rows in alphabetical order. If you specify nothing, that is, do not use a storage map to place rows of the table into the storage area using the sorted index, the rows are stored randomly.

When you define an index, you can specify a list of columns within the index that Oracle Rdb should use as a key for determining in which storage area to place an index. You can set the maximum value for the key when it is initially stored in a specified storage area by using the WITH LIMIT OF option in the STORE USING clause of the SQL CREATE INDEX statement. The column names specified in the STORE USING clause in the list must be in the same order as they appear in the index. When you define the storage map, you name the index to be used for placing rows in the storage area. If you are storing rows in multiple storage areas, you set the maximum value for the key when it is initially stored in a specified storage area by using the WITH LIMIT OF option in the STORE clause of the SQL CREATE STORAGE MAP statement. The columns in the list must be in the same order as they appear in the defined index. For the syntax and more information about this statement, see the *Oracle Rdb7 Guide to Database Design and Definition* and the *Oracle Rdb7 SQL Reference Manual*.

A cluster index can result in a significant difference in performance for queries requiring range retrievals. Suppose that you have defined a table to be clustered using an index based on the LAST_NAME column, and that you write a query that asks for all employees whose last name is greater than Smith. If there are a hundred employees whose last names are greater than Smith and five employees fit on a page, it takes Oracle Rdb approximately 20 I/O operations to answer the query with a clustering index. With a nonclustering index, Oracle Rdb may take as many as 100 I/O operations.

3.9.6.4 Preventing Performance Degradation That May Occur over Time Using Sorted Indexes

When a sorted placement index is used to store rows in a storage area, each row is stored in index key sequence near its pointer record in the index node record. Storage begins at the first available page in the storage area immediately following the space area management (SPAM) page. As each page is filled, the next available page in sequence is used to store rows, and so forth. Once an index node record becomes full, a new index node record is written on the next available page. A forward pointer is written in the original node record, chaining it to the new index node record. Thus, each row is stored in the storage area in index key sequence in close proximity to its pointer record, starting at the beginning of the storage area and continuing until all table rows and index node records are stored.

After a data load operation, range retrieval performance is optimal because rows and their pointer records are as closely placed to one another as possible under the default or specified tuning parameters. Range retrieval of rows is possible in one to a few I/O operations assuming that page size, index parameters such as node size and percent fill, and buffer sizes are tuned to optimize retrieval performance for the application.

It is only after new rows are inserted in the database or many updates are made to index key column data values following an initial data load operation that you can expect range retrieval performance to slowly degrade. The reason is simple. New rows are stored on the next available page with space, usually after the last full page in the storage area.

Index row pointer entries, however, are inserted in each respective index node record that should contain the new index key column value. This results in a row and index pointer record page displacement greater than what normally occurs under an initial data load operation. Furthermore, as row pointers are added to index nodes, node records are forced to split and rebalance. New node records are then written on the next page after the last full page in the storage area, causing additional displacement from existing rows. As rows are updated, rows never move but their pointer records are updated and relocated if the index key column value for the rows changes; the pointer record is relocated to the appropriate node record to maintain index key sequence. Again, as node records split and rebalance, rows and their node record pointers can be displaced further and further from each other. Therefore, page displacement is possible with the addition of new rows or update of existing rows.

This physical page displacement of each node pointer record from its data row leads to a degradation in retrieval performance. Over time, with the addition of more new data and changed key index column values, more I/O operations are needed to bring a specified range retrieval of index key values and associated rows into a buffer. Eventually, performance can be poor.

To prevent this problem, you should occasionally unload and reload the data in the storage area to optimally minimize the page displacement of each index node record pointer with its row. Prior to reloading the data, reexamine index and storage space parameters to see if further tuning is possible to optimize retrieval performance.

3.9.6.5 Forward and Reverse Scans Using a Sorted Index

Oracle Rdb can perform forward and reverse scans on a single sorted index, so that data can be retrieved in its standard sorted order or the reverse order.

When a reverse scan is returning dbkeys and index keys for a set of duplicate records (records with the same index key value), these duplicate record dbkeys are returned in the same order as they would be by a forward scan. This greatly improves the efficiency of the reverse scan of duplicate keys.

This optimization is also semantically acceptable because a sorted index does not guarantee any particular ordering among records whose index key values are identical.

Although two indexes are not required to perform index scans in both a forward and reverse direction, a reverse scan may use more CPU resources than a natural forward scan.

In Example 3–50, RDMS$DEBUG_FLAGS has been set to "S" so that the query strategy selected by the optimizer is displayed as the query executes. Note that the RDMS$DEBUG_FLAGS output for the second statement shows that a reverse scan is used to process the query.

Example 3–50 Using a Single Sorted Index for Forward and Reverse Scans

```
SQL> -- The optimizer uses forward scan for the LAST_NAME index
SQL> -- (an ascending index) when the values are requested in the
SQL> -- same order as the index (ascending).
SQL> SELECT LAST_NAME, FIRST_NAME FROM EMPLOYEES
cont> WHERE LAST_NAME > 'A' AND LAST_NAME < 'M'
SQL> ORDER BY LAST_NAME ASCENDING;
```

(continued on next page)

Example 3–50 (Cont.) Using a Single Sorted Index for Forward and Reverse Scans

```
Conjunct        Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_LAST_NAME [1:1]
 LAST_NAME        FIRST_NAME
 Ames             Louie
 Andriola         Leslie
   .
   .
   .
 Lengyel          Peter
 Lonergan         Peter
57 rows selected
SQL> --
SQL> -- When the data is requested in the reverse order of the
SQL> -- ascending EMP_LAST_NAME index in the following statement,
SQL> -- the optimizer uses a reverse scan of the index to retrieve
SQL> -- the values.
SQL> SELECT LAST_NAME, FIRST_NAME FROM EMPLOYEES
cont> WHERE LAST_NAME > ’A’ AND LAST_NAME < ’M’
SQL> ORDER BY LAST_NAME DESCENDING;
Conjunct        Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_LAST_NAME [1:1]       Reverse Scan
 LAST_NAME        FIRST_NAME
 Lonergan         Peter
 Lengyel          Peter
   .
   .
   .
 Andriola         Leslie
 Ames             Louie
57 rows selected
SQL>
```

Note that you cannot specifically request as part of your query that a forward scan or a reverse scan of a particular index be used to process the query. In Example 3–50, the optimizer determined that a forward scan and reverse scan of the sorted index on the LAST_NAME column was the fastest way to process the queries. With other queries that specify a sorted index in the ORDER BY clause, the optimizer might find a more efficient method of processing the query. If you want the optimizer to use a particular index when processing a query, you can define a query outline that specifies the index you want the optimizer to use when processing the query. See Section 5.9 for details.

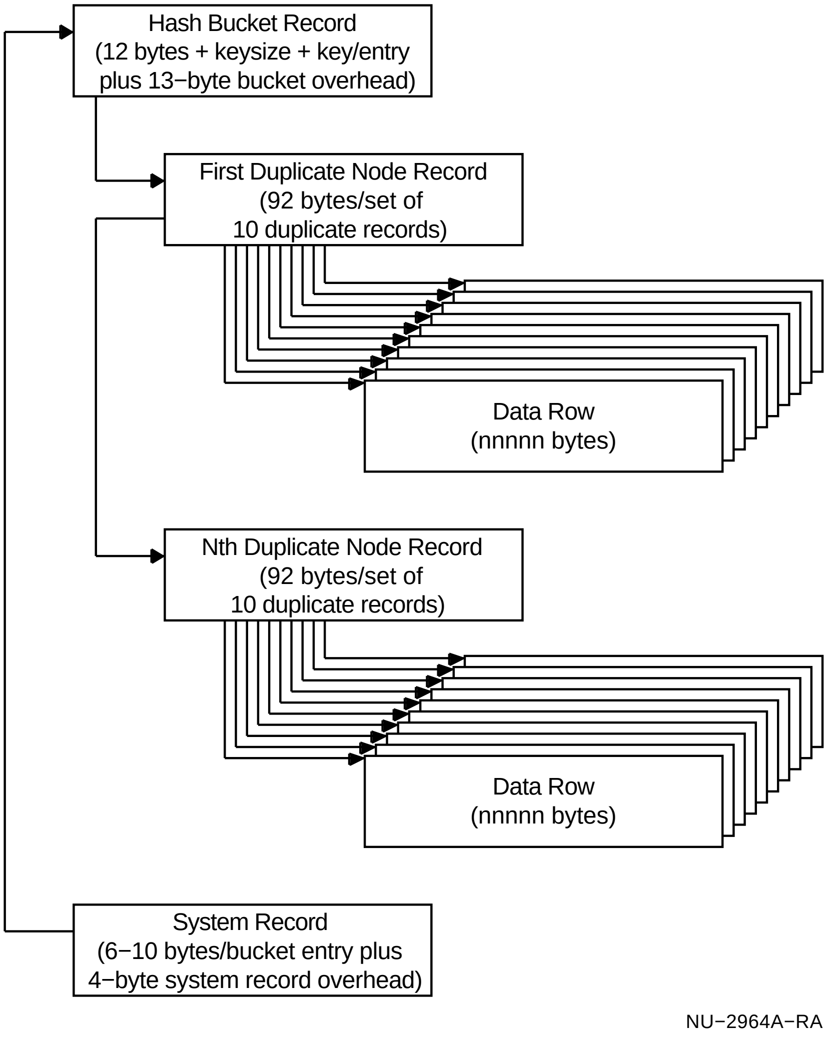
3.9.7 Hashed Index Structure

The hashed index is made up of the following three types of records:

- The system record that contains one pointer for each hash bucket on the page
- The hash bucket record that contains pointers to the data rows or if duplicate records are allowed, pointers to the duplicate node records
- The duplicate node record that contains one pointer to each duplicate data row, holds a maximum of 10 pointers, and if there are more duplicate data rows, contains a pointer to the next duplicate node record

Figure 3–11 shows how the three types of hashed index records are related to the data row when both the hashed index and data rows are stored in the same storage area. Lengths of each record are indicated in parentheses.

Figure 3–11 Hashed Index Structure


Hash Bucket Record
(12 bytes + keysize + key/entry
plus 13−byte bucket overhead)
First Duplicate Node Record
(92 bytes/set of
10 duplicate records)
Data Row
(nnnnn bytes)
Nth Duplicate Node Record
(92 bytes/set of
10 duplicate records)
Data Row
(nnnnn bytes)
System Record
(6−10 bytes/bucket entry plus
4−byte system record overhead)
NU−2964A−RA

For further discussion of each record type, see the *Oracle Rdb7 Guide to Database Maintenance.*

3.9.7.1 Hashed Index Performance Factors

This section discusses hashed index performance. For additional information, refer to Section 8.1.3.5.

When you define a hashed index, the following factors and database parameters are important:

- Record size
- Page size
- Initial allocation
- Page format—mixed page format only
- SPAM thresholds
- SPAM interval
- Key size
- Estimate of the number of unique key values
- Estimate of the number of duplicate rows
- Estimate of the total number of rows in the table associated with the hashed index
- Where the hashed index is stored relative to the data—in separate storage areas or in the same storage area using the PLACEMENT VIA INDEX clause

A hashed index retrieves exact matches to the search key in as few as two I/O operations when both data and hashed index reside in different storage areas, or an average of one I/O operation when both data and hashed index occur in close proximity to each other within the same storage area. It is best to do some size calculations and understand your data, especially how your duplicate rows are distributed, to estimate how performance might be improved using a hashed index for your tables. Size estimation is explained in detail in the *Oracle Rdb7 Guide to Database Design and Definition*.

3.9.7.2 Potential Sizing Problems

For the hashing algorithm to operate with greatest efficiency when placing rows using the PLACEMENT VIA INDEX clause, sufficient space must be allocated to ensure that rows are distributed randomly and uniformly across all data pages in the storage area. When rows begin filling data pages in a non-uniform way, perhaps due to insufficient page size, the efficiency of the hashed index slowly declines until performance degradation may be noticeable. This is true because the hashing algorithm formulates the dbkey from the search key. If the page is full, the next nearest adjacent page is scanned for empty space using the SPAM thresholds you set for the storage area as a guide. This in turn causes problems when that page is chosen as a target. If this page is full, the next page in sequence is scanned. The first available page with sufficient space becomes the new location for the row.

Over time, as rows are placed using the hashed index, performance may degrade to the point where queries that use this hashed index are affected and some action must be taken. A hash retrieval query on the search key picks up the dbkey and locates the page where the row should be. But a check of the system record for that page may indicate that the hash bucket is located 10 pages away. The dbkey of the duplicate node record may indicate that it is located on that same page, but the second duplicate node record is an additional 60 pages away and the data row is another 30 pages away from it, for a total displacement of 100 pages from the system record or perhaps three I/O operations.

The result is poorer performance than expected because of the duplicate record problem compounded with too small a page size and buffer size to handle all those duplicate rows. Use the RMU Analyze command to detect these problems.

To solve some of these problems, you should recalculate values for the important parameters mentioned in the summary list in Section 3.9.7.1 and do one of the following:

- Define new storage areas that specify the new calculated values. Modify the associated storage maps to point to these new storage areas. Load your data into the new storage areas by using the REORGANIZE clause of the SQL ALTER STORAGE MAP statement. Delete all empty unused storage areas.
- Unload the data from the storage areas using the RMU Unload command. Define new storage areas that specify the new calculated values. Delete the old storage areas. Load the data again into these new storage areas by using the RMU Load command.
- If changes are extensive and you want to perform them all in one operation, use the SQL EXPORT and IMPORT statements to initiate these desired changes.

Using a hashed index for retrieval of exact matches to the search key for tables with large numbers of unique keys has great benefit and little risk. Using a hashed index for updates, like using the sorted index, is also of great benefit, but with some risk if you do not make accurate estimates of these important parameters: storage area allocation, page size, SPAM thresholds, SPAM interval, and values for the parameters used for calculating the size of a hashed index and data area. As your database grows, it is important to know what aspects of it are growing the fastest to anticipate problems.

3.9.7.3 Shadow Pages

When it is not practical or is physically impossible to cluster many different table rows and hashed indexes on the same page to gain optimum performance, then the next best alternative is to use two different storage areas to store related tables and use shadow pages where a minimum of two I/O operations is guaranteed. **Shadowing** is the placement of child rows in the same relative offset of another storage area that is apart from the related parent rows. Example 3–51 shows the index and storage map definitions and how to shadow the JOB_HISTORY rows with the EMPLOYEE rows in two different storage areas. Parent rows and both hashed indexes are stored in one storage area and the child rows are stored in a second storage area. Both parent and child data rows are placed using the PLACEMENT VIA INDEX clause.

Example 3–51 Shadowing Achieves a Clustering Effect Between Two Different Storage Areas for Storing and Retrieving Parent/Child Data Rows

```
CREATE UNIQUE INDEX EMPLOYEE_HASH      CREATE INDEX JOB_HISTORY_HASH
  ON EMPLOYEES (EMPLOYEE_ID)             ON JOB_HISTORY (EMPLOYEE_ID)
  STORE IN AREA_B                        STORE IN AREA_B
  TYPE IS HASHED;                        TYPE IS HASHED;

CREATE STORAGE MAP EMP_MAP              CREATE STORAGE MAP JOB_HISTORY_MAP
  FOR EMPLOYEES                           FOR JOB_HISTORY
  STORE IN AREA_B                         STORE IN AREA_A
  PLACEMENT VIA INDEX EMPLOYEE_HASH;      PLACEMENT VIA INDEX JOB_HISTORY_HASH;
```

Shadowing is a good strategy when you have a small amount of data, as in the EMPLOYEES table, with possibly a large number of related child rows in the JOB_HISTORY table, inserted randomly over a period of time. Shadowing ensures that the transactions will be grouped together, and if enough space is allocated in AREA_B, then free space will be available to maintain the clustering effect.

Shadowing works in the following way: JOB_HISTORY shadow rows in storage area AREA_A are stored at the same relative offset in the storage area but not necessarily on the same page numbers as the related EMPLOYEES rows stored in storage area AREA_B. The grouping allows better utilization of the in-memory buffers. When you read a buffer from storage area AREA_B to get the hashed index bucket for the EMPLOYEES_HASH index, the buffer from storage area AREA_B already has the related EMPLOYEE rows in it. The buffer also includes the hash bucket for the JOB_HISTORY_HASH index that contains the pointers to the related JOB_HISTORY data rows in storage area AREA_A. Access by hashed indexes for both the EMPLOYEES and JOB_HISTORY tables therefore requires two I/O operations, one to access AREA_B for the JOB_HISTORY hashed index, and one to access the JOB_HISTORY data row in AREA_A. Also, if AREA_A is a uniform area, then sequential access to JOB_HISTORY rows will perform much better with the rows still distributed similarly to the parent rows.

For the syntax and more information about the PLACEMENT VIA INDEX clause, see the *Oracle Rdb7 SQL Reference Manual*.

3.9.8 Selecting the HASHED SCATTERED or HASHED ORDERED Hashing Algorithms

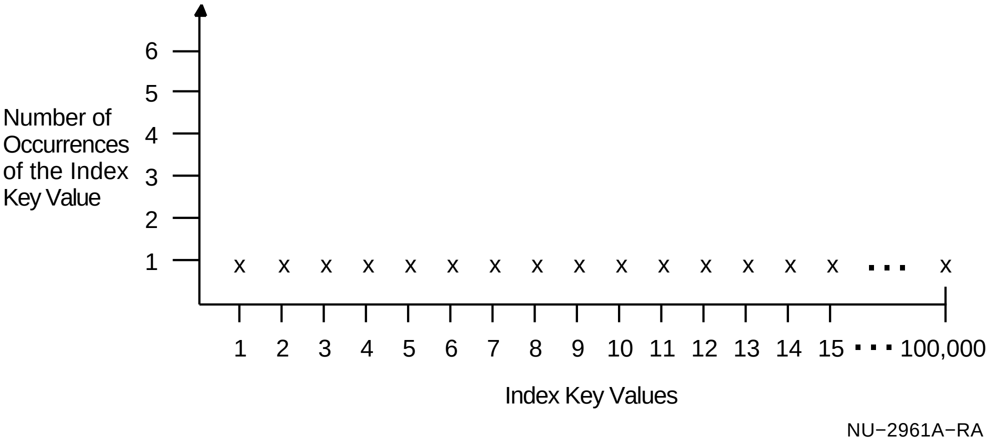
When you define a new hashed index, you can select which of two available hashing algorithms Oracle Rdb will use to store index keys for the index. You can select the TYPE IS HASHED SCATTERED or TYPE IS HASHED ORDERED clause of the SQL CREATE INDEX statement. The SCATTERED option is the default.

The HASHED SCATTERED hashing algorithm converts a multi-octet string into a single longword that is then mapped to the initial allocation of the storage area to determine a target page. One of the characteristics of this algorithm is the scattering of data across the area. For instance, two adjacent key values might, in fact, be widely separated in the storage area. If data is loaded in sorted order, this scattering has high random I/O cost; in this case, users should use the Place qualifier of the RMU Load command, or the PLACEMENT ONLY clause of the SQL INSERT statement to preprocess data before loading.

The record distribution pattern is not usually uniform with the HASHED SCATTERED algorithm; some pages are chosen as targets more often than others. Some pages might even remain empty while others receive multiple entries.

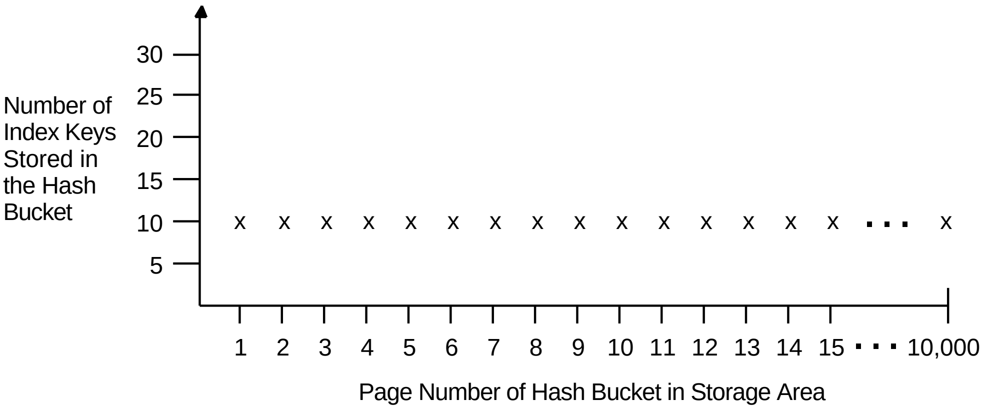
The HASHED ORDERED algorithm is ideal for applications in which the key values are uniformly distributed across a range. That is, the HASHED ORDERED algorithm should be used when an application has a range of index key values, and each key value occurs the same number of times. An application with a range of sequential index key values between 1 and 100,000 with no duplicate values would benefit from using the HASHED ORDERED algorithm. Figure 3–12 shows a graph of the range of index key values for this application. You should not consider specifying the HASHED ORDERED clause for an application unless a plotting of the index key values for the application produces a horizontal (or nearly horizontal) line on the graph shown in Figure 3–12.

Figure 3–12 Identifying Index Key Values That Are Appropriate for the HASHED ORDERED Option



The HASHED ORDERED algorithm uses the first four octets of each key to distribute the data uniformly across the storage area. Figure 3–13 shows the uniform distribution that results when the index key values shown in Figure 3–12 are stored using the HASHED ORDERED algorithm. Notice that each hash bucket contains the same number of index keys; when the storage pattern for the index keys is plotted on the graph, the result is a horizontal line like the one in Figure 3–12. This shows that when index key values are uniformly distributed across a range, the HASHED ORDERED algorithm distributes these values uniformly across a storage area.

Figure 3–13 Distribution of Data Across a Storage Area When the HASHED ORDERED Option Is Used with Appropriate Index Key Values



NU−2962A−RA

You should use the HASHED ORDERED option only when all of the following are true:

- The index keys are integer values, dates, timestamps, or intervals.

  Note

  In the BIGINT, DATE (both ANSI and VMS), TIME, TIMESTAMP, and INTERVAL data types, the hash algorithm uses only the low order longword.

  The fractional precision (scale) is ignored for the purposes of this hash algorithm. For example, consider the values of 1.1 and 2.1 for a column specified as INTEGER(1). Values 1.1 and 2.1 are considered identical to 11 and 21 and so will not be on adjacent database pages as might be expected.

- The index is defined as an ASCENDING index
- There is a range of index key values, and each key value occurs the same number of times
- The MAPPING VALUES clause was not used when defining the index (so the index does not compress all numeric index key values)

A hashed ordered index can reference more than one column. The last column in the list must conform to the data type restrictions in the preceding list, and is the only part of the index used for distribution of data within a storage area. However, all other columns in the list may be used in a STORE USING . . . WITH LIMIT clause to partition the data first between different storage areas.

The following example shows a hashed ordered index referencing more than one column:

```
SQL> -- A company has 4 warehouses and within each warehouse are 10000
SQL> -- products.  The warehouse managers want the data for each warehouse
SQL> -- to be stored in separate storage areas with the stock rows evenly
SQL> -- distributed within each area.
SQL> --
SQL> CREATE TABLE STOCK_CONTROL
cont>   (WAREHOUSE    INTEGER,
cont>    STOCK_NO     INTEGER,
cont>    DESCRIPTION  CHAR(20) );
SQL> --
SQL> CREATE STORAGE MAP STOCK_MAP
cont>   FOR STOCK_CONTROL
cont>   STORE IN AREA_W0;
SQL> --
SQL> -- Use WAREHOUSE to partition across disks.
SQL> -- Use STOCK_NO to order the stock numbers across the area.
SQL> --
SQL> CREATE UNIQUE INDEX STOCK_INDEX
cont>   ON STOCK_CONTROL (WAREHOUSE, STOCK_NO)
cont>   TYPE IS HASHED ORDERED
cont>   STORE USING (WAREHOUSE)
cont>     IN AREA_W1 WITH LIMIT OF (1)
cont>     IN AREA_W2 WITH LIMIT OF (2)
cont>     IN AREA_W3 WITH LIMIT OF (3)
cont>     IN AREA_W4 WITH LIMIT OF (4);
SQL> COMMIT;
```

If you have a set of uniformly distributed index key values and you need to determine the initial size of the storage area that will hold the records associated with the key values, you can reduce the amount of overhead for the data pages in the area by using the HASHED ORDERED algorithm for the index. Suppose, for example, that you have a set of sequential index key values between 1 and 100,000 with no duplicate values and you want to define a storage area to hold this data. If you know that 10 records containing these index key values can fit on a page, you can define a storage area with an initial allocation of 10,000 data pages (10,000 pages times 10 records per page equals 100,000 records). If you use the HASHED SCATTERED algorithm to store these records after defining the new storage area, an average of 10 records will be stored per page, but the number of records will vary from page to page.

Therefore, you need to provide some overhead for each page in the storage area because the HASHED SCATTERED algorithm will store more than 10 records on some pages. However, when the HASHED ORDERED algorithm is used to store the records, exactly 10 records will be stored on each data page. Thus, you can reduce the amount of overhead per page when you use the HASHED ORDERED algorithm, which reduces the total size of the storage area and saves disk space.

3.9.9 Sequential Retrieval

How Oracle Rdb chooses a sequential access strategy for a query depends on many factors. One factor is whether or not any indexes are defined for columns specified in the record selection expression (RSE). Or, the optimizer may decide that a sequential search is the most efficient. Whatever the reason, sequential access to rows in a table does not take advantage of adjustable locking.

Because sequential retrieval requires each row in the table to be tested to determine if it satisfies the query's RSE, Oracle Rdb increases (or promotes) the level of the lock on all rows *accessed* by the query to keep them stable. Otherwise, other transactions could intervene and modify rows during the sequential search. Therefore, sequential retrieval for a read/write transaction has the following characteristics:

- Requires locks on the entire table, which results in coarser locks.
- Uses a protected read mode on the table to prevent any subsequent update transactions from modifying rows during the current transaction. The protected read mode does not allow update transactions to access the read-protected resources at the same time.

Refer to Section 4.1.12 and Section 4.1.13 for information about the effect of snapshot files on read-only transactions.

Because adjustable locking is not in effect during a sequential search, Oracle Rdb secures all the rows in the table from access by other transactions. For this reason, try to avoid the use of sequential searches. When a transaction that has reserved a table in shared write mode accesses a table sequentially, as shown in Example 3–52, a protected write lock is placed on the table. This denies access to that database resource by all subsequent read/write transactions except for shared read transactions.

For example, suppose you need to access the EMPLOYEES table using a column that does not have an index defined for it. Because the optimizer does not have the option of using an index to locate rows directly, Oracle Rdb must sequentially search the table itself until the rows identified by the RSE are found. Because processing is sequential, the whole table is locked during the search for the duration of the transaction.

Example 3–52 Sequential Access of the EMPLOYEES Table

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SELECT E.LAST_NAME, E.SEX, E.CITY
cont> FROM EMPLOYEES E
cont> WHERE E.POSTAL_CODE BETWEEN "03103" AND "03601"
cont> AND E.STATE = "NH";
SQL>
```

But if you define an index for both the POSTAL_CODE and STATE columns, you gain the advantage of adjustable locking. Consequently, the current transaction locks only those rows that satisfy the RSE and the index nodes associated with them (locks all rows accessed). Adjustable locking allows other transactions to select from the remaining rows in the table without interfering with the current transaction. For example, if there is only an index on the STATE column, the query shown in Example 3–53 locks all the rows with the value NH, even those rows not in the specified POSTAL_CODE range.

Example 3–53 Indexed Access and Adjustable Locking

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SELECT E.LAST_NAME, E.SEX, E.CITY
cont> FROM EMPLOYEES E
cont> WHERE E.POSTAL_CODE BETWEEN "03103" AND "03601"
cont> OR E.STATE = "NH";
SQL>
```

3.10 Recognizing Poor Insert Performance Caused by Excessive Page Checking

You may encounter poor performance when you store new rows in a database. Sometimes this is due to the way that Oracle Rdb manages free space on data pages. Sometimes Oracle Rdb reads many pages in a storage area before it finds a page with sufficient space on it to store a row. This section describes the symptoms you may see and presents ways to prevent poor insert performance that can occur in the situations described.

The simplest way to verify if any of the scenarios in this section are occurring is to use the Record Statistics and the File IO Overview displays of the Performance Monitor. The Record Statistics display provides two entries of particular interest:

- pages checked
- discarded

If the values for the discarded field are significant in comparison with the pages checked values, then you can investigate further. Section 3.10.1 describes how to use the File IO Overview display to identify excessive IO in storage areas.

Another way to validate this is to watch the Stall Messages screen in the Performance Monitor while a process is experiencing the slow insert performance. Look for the message “reading pages n:n to n:n” for that particular process. If you see it persistently showing up and the page numbers displayed steadily increase, then it is likely that the process is searching for a database page that has sufficient space to store a row. Note that this stall message is also displayed if Oracle Rdb is sequentially reading a table; you should verify that the process is doing an insert before assuming that this stall message indicates a problem with excessive page checking. The statistic “pages checked” should also be steadily increasing at the same time.

Oracle Rdb uses space area management (SPAM) pages to determine what database pages have space available for new rows. The SPAM page is essentially a table representing a range of pages in a storage area. There is an entry in the table for each page in the range, and each entry shows whether or not space is available on the respective page. When searching for a page to store a row, Oracle Rdb searches for SPAM entries that show that a page has sufficient space available. There are conditions, however, where SPAM pages may not reflect the actual availability of space on a data page. When Oracle Rdb finds an entry that the SPAM page shows has space available on it, the page is retrieved and the actual available space on the page is examined to confirm that there is sufficient free space to store the new row. Sometimes the available space on the page may not actually be enough to store the row. Those cases are listed below:

- Thresholds are set incorrectly on a mixed format storage area or on logical area thresholds defined for a table or index.
- Some or all of the free space is locked for use by other database users.
- The length used to represent the size of the data rows does not represent the actual length of the data rows. This is always the case for duplicate nodes on indexes that allow duplicates.

3.10.1 Identifying Excessive I/O in Storage Areas

After you detect the problem that Oracle Rdb is checking excessive pages while attempting to insert a row, you can use the File IO Overview screen to identify which storage area is exhibiting the behavior. The cause of the problem could be locked free space, incorrect SPAM thresholds, unique indexes, or estimated record sizes.

For high volume, transaction processing applications or applications with a large number of storage areas, it is not practical to manually examine the IO Statistics (by file) screen for each storage area in the database, trying to identify the particular storage area with excessive read I/O operations.

To help you more quickly identify the storage area or areas in which excessive page checks are occurring, use the File IO Overview screen from the IO Statistics (by file) submenu.

The File IO Overview screen shows the synchronous and asynchronous read and write I/O counts, and the database pages checked for all storage areas, the .aij file, the .ruj file, the .ace file, the database root file and, finally, all data and snapshot areas combined.

For example, consider the following File IO Overview screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 19-JUN-1996 10:21:30
Rate: 0.50 Seconds      File IO Overview (Total Checked)    Elapsed: 03:39:10.39
Page: 1 of 9       KODD$:[R_ANDERSON.WORK.AIJ]OE_RDB.RDB;1        Mode: Online
--------------------------------------------------------------------------------
File/Storage.Area.Name........ Sync.Reads SyncWrites AsyncReads AsyncWrits PgCkd
All data/snap files                198752      23622      21388      57085  597k
data OE$ORDER1                        566        791       7103       2399  164k
data OE$ORDER10                       572        849       7103       2374  164k
data OE$ORDER4                        567        916       7177       2287  164k
data OE$CUSTOMER1                   15676       3060          0      11936 19096
data OE$CUSTOMER10                  15713       3158          0      11820 19096
data OE$CUSTOMER4                   15608       3633          0      11367 19096
data OE$STOCK04                      2027       4338          5        524 12136
data OE$STOCK01                      2131       3506          0       1254 11854
data OE$ITEM1                      132913        754          0       1938  6011
data OE$NEW_ORDER1                   3869        597          0       3808  5310
data OE$NEW_ORDER4                   3818        788          0       3599  5308
data OE$NEW_ORDER10                  3898        679          0       3732  5306
data OE$WAREHOUSE1                     11          3          0          8    14
data OE$WAREHOUSE10                    11          4          0          7    14
data OE$WAREHOUSE4                     11          5          0          6    14
ACE (AIJ Cache Electronic)              0          0          0          0     0
--------------------------------------------------------------------------------
Config Exit Help Menu >next_page <prev_page Options Reset Set_rate Unreset Write
```

This example shows a File IO Overview screen that has been configured to display the synchronous and asynchronous read and write counts, sorted by descending total pages checked. Database storage areas are identified by the prefix *data* and snapshot areas are identified by the prefix *snap*. Storage areas that are added to the database are automatically shown on the screen.

The “PgCkd” column indicates the number of database pages checked for each area, and summarizes this information for all data/snap areas.

You can configure the File IO Overview screen to sort the storage area information in several different ways. Type C to select the Config option from the screen’s horizontal menu, which displays the configuration menu:

```
+------ Select Display Configuration -----+
|                                         |
|  A.  Unsorted totals display            |
|  B.  Sort by total synchronous reads    |
|  C.  Sort by total synchronous writes   |
|  D.  Sort by total asynchronous reads   |
|  E.  Sort by total asynchronous writes  |
|  F.  Sort by total reads & writes       |
|  G.  Sort by total pages checked        |
|  H.  Unsorted current rate display      |
|  I.  Sort by synchronous read rate      |
|  J.  Sort by synchronous write rate     |
|  K.  Sort by asynchronous read rate     |
|  L.  Sort by asynchronous write rate    |
|  M.  Sort by total current I/O rates    |
|  N.  Sort by pages checked rates        |
+-----------------------------------------+
```

When you select options B through N, storage areas with duplicate sort criteria are displayed alphabetically.

The File IO Overview screen name in the display header shows the display configuration you selected.

The File IO Overview screen is not available when the Input command line qualifier is used to replay an output data file.

See the Performance Monitor help for more information about the configuration options.

3.10.2 Incorrect Threshold Settings

Suppose you observe that excessive pages are checked when rows are stored in a table under one of the following conditions:

- The table is stored in a mixed format area
- The table has logical area thresholds defined in its storage map
- The table has indexes defined that contain logical area thresholds

The excessive page checking may be caused by incorrect threshold settings. Review the calculations used to define the thresholds to verify that the thresholds have been set correctly.

3.10.3 Locked Free Space

When a process deletes rows from a database, and the dbkey scope is set for the duration of an attachment to the database, the space that was allocated to the deleted rows is reserved or "locked" by that process until the process detaches from the database. This is done to ensure that the rows can be put back on the page if the transaction is rolled back. At the time the rows are deleted, the SPAM page is updated to reflect the space that was made available by deleting the rows. When other database processes begin searching for available space to store rows, they see in the SPAM page that there may be space available on the page where rows were deleted. But, when that page is actually retrieved, Oracle Rdb finds that some or all of the available space on the page is locked for another process and insufficient space exists to store another row. Another page must be selected for storing the row, and the preceding process repeats.

Typically this scenario has little impact on database performance, but in situations where, for example, a long-running database maintenance process that deletes rows on many pages is active, many database pages may contain locked free space. This can significantly impact performance if many processes are attempting to store rows in the same area at the same time the maintenance process is running.

3.10.4 Stored Values for AIP Lengths May Reflect the Actual Length of Table Rows

Oracle Rdb stores the nominal length of table rows in a structure called the area inventory page (AIP). The length stored in the AIP is used to determine how much space must be available before a page in a uniform format area is considered full. The length stored in the AIP for a row is *not* the actual length of the row in the following three circumstances:

- Indexes that allow duplicates

- Unique indexes and tables that have been altered and had columns added or dropped
- Segmented strings (LIST OF BYTE VARYING data type)

3.10.4.1 AIP Length Problems in Indexes That Allow Duplicates

When an index allows duplicates, the length stored in the AIP will be 215 bytes, regardless of the actual index node size. Because an index with duplicates can have variable node sizes, the 215-byte size is used as a median length to represent the length of rows in the index's logical area. When the row size in the AIP is less than the actual row length, it is highly likely that SPAM entries will show space is available on pages when they have insufficient space to store another full size row. This is the most common cause of insert performance problems.

For example, consider a case where an index node size of 430 bytes (a common default value) is used; the page size for the storage area where the index is stored is 2 blocks. After deducting page overhead, the available space on a 2-block page is 982 bytes. Assume that the page in this example is initially empty.

1. A full size (430-byte) index node is stored. As 8 bytes of overhead are associated with each row stored on a page, that leaves 982 – 430 – 8 = 544 free bytes remaining on the page.
2. A duplicate key entry is made in that index node and thus a duplicate node is created on the same page. An initial duplicate node is 112 bytes long (duplicate nodes can have a variety of sizes depending on when they are created, but for this particular example, 112 bytes is used). Therefore, 544 – 112 – 8 = 424 free bytes remain on the page.

At this point, 424 bytes are left on the page. That is greater than the 215 bytes that the AIP shows as the row length for the logical area, so the SPAM page shows that the page has space available. However, an attempt to store a full size index node on the page will fail, because the remaining free space (424 bytes) is not enough to store a 430-byte node.

In this case, another candidate page must be selected via the SPAM page, and the process repeats until a page that truly has sufficient free space available is found. In a logical area that contains many duplicate nodes, a significant percentage of the pages in the logical area may fit the scenario just described. When that is the case, and a new full size index node needs to be stored, many pages may need to be read and checked before one is found that can be used to store the row.

It is possible to avoid the preceding scenario by using logical area thresholds. The goal is to set a threshold such that the SPAM page will show a page is full when space is insufficient to store a full size index node.

Using the previous example, here is how to properly set logical area thresholds to prevent excessive pages checked on an index with a 430-byte node size that is stored on a 2-block page. To calculate the proper threshold value to use, you must first determine how full the page can get before no more full size nodes will fit on the page. In this example, a database page can have up to 982 – 430 – 8 = 544 bytes in use before the page is too full. Therefore, if 544 or fewer bytes are in use, then enough space remains to store another full size node. The threshold is then 544 / 982 = .553971, or 55%.

Here is an example of creating an index with the above characteristics:

```
SQL> CREATE INDEX TEST_INDEX ON EMPLOYEES (LAST_NAME)
cont>     STORE IN RDB$SYSTEM
cont>     (THRESHOLD IS (55));
```

Note that the compression algorithm used on regular tables that have compression enabled does not apply to index nodes. Index nodes are not compressed like data rows and will always utilize the number of bytes that is specified in the node size. Do not attempt to take into account a compression factor when calculating thresholds for indexes.

3.10.4.2 AIP Length Problems in Segmented Strings

The length stored in the AIP for segmented strings is always 150 bytes, regardless of the actual size of the segmented string. Thus, the same problems described in Section 3.10.4.1 also apply to segmented strings. If a page in a uniform area has 158 (150 + 8 overhead bytes = 158) or more bytes free on it, then the SPAM entry for that page will show that space is available. However, if an attempt is made to store a segmented string that is larger than the actual free space on that page, the page will be checked when it does not have sufficient free space to store the segment.

The proper way to avoid this condition is to store segmented strings in a mixed format storage area only. The storage area should have appropriate thresholds defined for the segment sizes being stored.

4

Adjusting Parameters

This chapter describes the database parameters and the OpenVMS system parameters that you can adjust to improve database performance. It provides guidelines to help you determine if you should enable selected Oracle Rdb features, such as global buffering, and recommends parameter values for those features that you do enable.

4.1 Adjusting Database Parameters

Table 4–1 summarizes the minimum effort needed to implement a physical design of your application, based on the kind of database (simple or complex) and the size of the database (small to very large). It describes how far the Oracle Rdb default values will take you before you need to consider changing them.

Table 4–1 How Far Will the Oracle Rdb Default Values Take You?

	Amount of Tuning	
Database Size	Simple Database: Less than 10–15 Tables	Complex Database: Fifteen or More Tables
Less than 10 Mb	None	Minimal
10 to 100 Mb	Minimal	Moderate
More than 100 Mb to 1 Gb	Moderate	Extensive
More than 1 Gb	Extensive	Extensive

Key to Tuning Effort

None—Practically no effort; single-file database, uniform page format, sorted indexes; accept most if not all default values.
Minimal—Minimal effort; probably single-file or maybe multifile, indexes defined properly, set buffers properly; change some default values.
Moderate—Moderate effort; multifile database, maybe mixed page format, maybe using hashed indexes, using segmented strings in mixed page format, set SPAM parameters; change many default values.
Extensive—Extensive effort; multifile; careful look at all features in combination, definitions; change most, if not all, default values; hashed indexes, B-tree node sizing.

Table 4–2 summarizes the Oracle Rdb default values for database-wide parameters.

Table 4–2 Oracle Rdb Database-Wide Parameter Values: Default, Minimum, and Maximum

Database-Wide Parameters	Default Values (Minimum/Maximum)
Description	None
OpenVMS VAX OpenVMS Alpha Path name	CDD$DEFAULT♦
Number of users	50 (1/2032)
Number of buffers	20 (2/32768)
Maximum number of cluster nodes	16 (1/96)
Number of recovery buffers	20 (2/32768)
Buffer size	3 times the maximum area page size

(continued on next page)

Table 4–2 (Cont.) Oracle Rdb Database-Wide Parameter Values: Default, Minimum, and Maximum

Database-Wide Parameters	Default Values (Minimum/Maximum)
Global buffers	Disabled
Row caching	Disabled
Fast commit processing	Disabled
Journal optimization	Disabled
Carry-over lock optimization	Enabled
Lock granularity	Enabled
Enable/disable snapshot file	Enabled
Snapshot immediate/deferred	Immediate
Database opening	Open is automatic
Specify after-image journal	AIJ is disabled until file name is specified
After-image journal allocation	512 blocks (0/disk device size)
After-image journal extent	512 blocks (no extent options) (0/disk device size)
OpenVMS VAX OpenVMS Alpha Requiring a dictionary	Dictionary is not required♦
OpenVMS VAX OpenVMS Alpha Using a dictionary	Dictionary is used♦

The following sections describe the database parameters and provide information you can use to determine optimum values for your database applications. Because you can often attain good performance with Oracle Rdb using the default values for each database parameter, you should use these sections as guidelines to adjust your database for applications with unusual characteristics.

Some characteristics of your application environment that influence database performance include the amount of virtual memory available, the number of users, and whether your applications are primarily read-only or update-intensive. Using the Performance Monitor screens discussed in Section 4.1.1, you can monitor database performance and data storage within the database file itself to determine if you need to adjust the database parameters. You can make any changes using SQL.

When your database was created, the database definer either accepted all of the Oracle Rdb default parameters or modified some of those parameters to suit site-specific needs. In most cases, the defaults Oracle Rdb uses provide good performance, particularly during the development phases of your database applications. Table 4–1 describes the applicability of the Oracle Rdb defaults relative to database size (number of rows) and database complexity (number of tables). If you need to modify existing database parameters, you can use the SQL ALTER DATABASE or IMPORT statement to specify new values for those database parameters you need to change. The chapter on modifying database and storage areas in the *Oracle Rdb7 Guide to Database Design and Definition* provides a table that shows which parameters can be updated while users are attached to the database, and which parameters can be updated only when users are not attached to the database.

Table 4–3 describes SQL statements that allow you to specify database-wide parameters.

Table 4–3 SQL Statements Affecting Database Parameters

Database-Wide Parameters	SQL CREATE DATABASE	SQL ALTER DATABASE	SQL IMPORT DATABASE
Description	Yes	No	Yes
Path name	Yes	No	Yes♦
Collating sequence is	Yes	No	Yes
Number of users	Yes	Yes (multifile only)	Yes
Number of buffers	Yes	Yes	Yes
Maximum number of cluster nodes	Yes	Yes (multifile only)	Yes
Number of recovery buffers	Yes	Yes	Yes
Buffer size	Yes	Yes	Yes
Enable/disable global buffers	Yes	Yes	No
Enable/disable row caching	Yes	Yes	No
Enable/disable fast commit	No	Yes	No
Journal optimization	No	Yes	No

OpenVMS VAX OpenVMS Alpha

(continued on next page)

Table 4–3 (Cont.) SQL Statements Affecting Database Parameters

Database-Wide Parameters	SQL CREATE DATABASE	SQL ALTER DATABASE	SQL IMPORT DATABASE
Enable/disable carry-over lock optimization	Yes	Yes	No
Lock granularity	Yes	Yes	Yes
Enable/disable snapshot file	Yes	Yes	Yes
Snapshot immediate/deferred	Yes	Yes	Yes
Database opening	No	Yes	No
Specify after-image journal	No	Yes	No
Specify no after-image journal	No (default)	Yes	No (default)
Journal allocation	No	Yes	No
Journal extent	No	Yes	No
Snapshot file allocation	Yes	Yes	Yes
Snapshot file extent	Yes	Yes	Yes
OpenVMS VAX OpenVMS Alpha Requiring a dictionary	Yes	Yes	Yes♦
OpenVMS VAX OpenVMS Alpha Using a dictionary	Yes	No	Yes♦
Segmented string storage area	Yes	No	Yes

Table 4–4 describes the Oracle RMU commands that allow you to specify database-wide parameters.

Table 4–4 Oracle RMU Commands Affecting Database Parameters

Database-Wide Parameters	RMU Restore	RMU Copy_ Database	RMU Move_Area
Description	No	No	No

(continued on next page)

Table 4–4 (Cont.) Oracle RMU Commands Affecting Database Parameters

OpenVMS VAX OpenVMS Alpha

Database-Wide Parameters	RMU Restore	RMU Copy_ Database	RMU Move_Area
Path name	Yes	No	No♦
Collating sequence is	No	No	No
Number of users	Yes	Yes	Yes
Number of buffers	Yes	No	No
Maximum number of cluster nodes	Yes	Yes	Yes
Number of recovery buffers	No	No	No
Buffer size	Yes	No	No
Row cache size	No	No	No
Enable/disable global buffers	Yes	No	No
Enable/disable fast commit	No	No	No
Journal optimization	No	No	No
Enable/disable carry-over lock optimization	No	No	No
Lock granularity	No	No	No
Enable/disable snapshot file	No	No	No
Snapshot immediate/deferred	No	No	No
Database opening	Yes	No	No
Specify after-image journal	Yes	Yes	Yes
Specify no after-image journal	Yes	Yes	Yes
Journal allocation	Yes	Yes	No
Journal extent	Yes	Yes	No
Snapshot file allocation	No	No	No
Snapshot file extent	No	No	No

(continued on next page)

Table 4–4 (Cont.) Oracle RMU Commands Affecting Database Parameters

	Database-Wide Parameters	RMU Restore	RMU Copy_ Database	RMU Move_Area
OpenVMS VAX OpenVMS Alpha	Requiring a dictionary	No	No	No♦
OpenVMS VAX OpenVMS Alpha	Using a dictionary	Yes	No	No♦
	Segmented string storage area	No	No	No

See the *Oracle Rdb7 Guide to Database Maintenance* and the *Oracle RMU Reference Manual* for more information on how to use the RMU Restore, the RMU Copy_Database, and the RMU Move_Area commands to modify specific database parameters.

4.1.1 Gathering Database Parameter Information

You can use the Performance Monitor screens, discussed in sections 4.1.1.1 through 4.1.1.12, to monitor database parameters.

4.1.1.1 Performance Monitor Database Parameter Information Submenu

The Performance Monitor provides a set of screens that display dynamic information. The information on these screens automatically changes to reflect database parameter modifications. When you select the Database Parameter Information option from the main menu, the following submenu is displayed:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1996 14:19:16
Rate: 3.00 Seconds          Summary IO Statistics          Elapsed: 03:58:22.56
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
------------------    +--------------------------------+
statistic.........    |         Select Display         |....... average......
name..............    |                                |....... per.trans....
transactions          |  A. General Information        |      7           1.0
verb successes        |  B. Buffer Information         |   4809         687.0
verb failures         |  C. Lock Information           |    241          34.4
                      |  D. Storage Area Information   |
synch data reads      |  E. Row Cache Information      |    259          37.0
synch data writes     |  F. Journaling Information     |      0           0.0
asynch data reads     |  G. Journal Information        |    105          15.0
asynch data writes    |  H. Fast Commit Information    |      0           0.0
RUJ file reads        |  I. Hot Standby Information    |      0           0.0
RUJ file writes       |  J. Audit Information          |      5           0.7
AIJ file reads        |  K. Active User Information    |      5           0.7
AIJ file writes       |  L. OpenVMS SYSGEN Parameters  |      0           0.0
ACE file reads        |                                |      0           0.0
ACE file writes       +--------------------------------+
root file reads                                              64           9.1
root file writes               0        0        0.0         31           4.4
--------------------------------------------------------------------------------
Type <return> or <letter> to select option, <control-Z> to cancel
```

The following example shows a Buffer Information screen:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1996 14:29:15
Rate: 3.00 Seconds             Buffer Information           Elapsed: 04:08:21.21
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------

Default user buffer count is 20
Default recovery buffer count is 20
Buffer size is 6 blocks
Global Buffers are enabled
- Global buffer count is 250
- Maximum global buffer count per user is 5
- Page transfer via memory is disabled
Global section size with global buffers disabled is 610007 bytes
- With global buffers enabled is 1522877 bytes
Asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 5 buffers
Detected asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 4 buffers
- Pre-fetch threshold is 4 pages
Asynchronous batch-write is enabled
- Clean buffer count is 5
- Maximum batch size is 4 buffers
--------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
```

The Database Parameter Information screens are not recorded in the binary output file produced using the Output qualifier. Consequently, these screens are not available when you replay a binary file using the Input qualifier.

See the Performance Monitor help for additional information on these screens.

4.1.1.2 Performance Monitor PIO Statistics–Data Writes Screen

The PIO Statistics–Data Writes screen provides information concerning data file writes and buffer unmarking activity (buffer writes to the disk). The following is an example of a PIO Statistics–Data Writes screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 17-MAY-1996 16:30:38
Rate: 3.00 Seconds        PIO Statistics--Data Writes       Elapsed: 00:00:39.01
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....
unmark buffer                22        0        9.7        12043          76.7
  transaction                 0        0        0.1           73           0.5
  pool overflow              22        0        9.4        11706          74.6
  blocking AST                0        0        0.0            0           0.0
  lock quota                  0        0        0.0            0           0.0
  lock conflict               0        0        0.1          165           1.1
  user unbind                 1        0        0.0           22           0.1
  batch rollback              0        0        0.0            0           0.0
  new area mode               0        0        0.0            0           0.0
  larea change                0        0        0.0           56           0.4
  incr backup                 0        0        0.0            1           0.0
  no AIJ access               0        0        0.0            0           0.0
  truncate snaps              0        0        0.0            0           0.0
  checkpoint                  0        0        0.0           20           0.1
  AIJ backup                  0        0        0.0            0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in this display, see the Performance Monitor help.

4.1.1.3 Performance Monitor PIO Statistics–Data Fetches Screen

The PIO Statistics–Data Fetches screen provides statistics on how data page requests are handled. There are two different formats for this screen: one for databases that have local buffers enabled, and another for databases that have global buffers enabled. The following is an example of a PIO Statistics–Data Fetches screen for a database with local buffers enabled:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 09:45:46
Rate: 3.00 Seconds       PIO Statistics--Data Fetches      Elapsed: 00:05:59.16
Page: 1 of 1      RDBVMS_USER1:[LOGAN.LOCAL]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

fetch for read              2287        0       20.4       18524         926.2
fetch for write               58        0        0.4         348          17.4

in LB: all ok               2035        0       18.3       16631         831.6
   LB: need lock             296        0        2.1        1949          97.5
   LB: old version             0        0        0.0           0           0.0

not found: read               34        0        0.3         292          14.6
         : synth               0        0        0.0           0           0.0

DAPF: success                  0        0        0.0           0           0.0
DAPF: failure                  0        0        0.0           0           0.0
DAPF: utilized                 0        0        0.0           0           0.0
DAPF: discarded                0        0        0.0           0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in this display, see the Performance Monitor help.

4.1.1.4 Performance Monitor PIO Statistics–SPAM Fetches Screen

The PIO Statistics–SPAM Fetches screen provides statistics on how SPAM page requests are handled. There are two different formats for this screen–one for databases that have local buffers enabled, and another for databases that have global buffers enabled. The following is an example of a PIO Statistics–SPAM Fetches screen for a database with global buffers enabled:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 10:02:33
Rate: 3.00 Seconds       PIO Statistics--SPAM Fetches      Elapsed: 00:00:24.49
Page: 1 of 1    RDBVMS_USER1:[LOGAN.GLOBAL]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....
fetch for read                 5        0        0.1           59           3.0
fetch for write                0        0        0.0            0           0.0
in AS: all ok                  4        0        0.0           52           2.6
   AS: lock for GB             0        0        0.0            0           0.0
   AS: need lock               0        0        0.0            0           0.0
   AS: old version             0        0        0.0            0           0.0
in GB: need lock               1        0        0.0            6           0.3
   GB: old version             0        0        0.0            0           0.0
   GB: transferred             0        0        0.0            0           0.0
not found: read                0        0        0.0            1           0.1
         : synth               0        0        0.0            0           0.0
DAPF: success                  0        0        0.0            0           0.0
DAPF: failure                  0        0        0.0            0           0.0
DAPF: utilized                 0        0        0.0            0           0.0
DAPF: discarded                0        0        0.0            0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in this display, see the Performance Monitor help.

4.1.1.5 Performance Monitor Asynchronous IO Statistics Screen

The Asynchronous IO Statistics screen provides information concerning asynchronous reads and writes to the database files. The stall times are displayed in hundredths of a second.

This screen shows the number of requests for asynchronous read and write operations, as well as the number of asynchronous read and write operations actually done.

You access the Asynchronous IO Statistics screen from the IO Statistics submenu. The following is an example of the Asynchronous IO Statistics screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 10:13:22
Rate: 3.00 Seconds        Asynchronous IO Statistics      Elapsed: 00:11:13.90
Page: 1 of 1        SQL_DISK1:[USER]MF_PERSONNEL.RDB;1             Mode: Online
--------------------------------------------------------------------------------

statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

data read request            67        0        6.3         2976        1488.0
data read IO                 14        0        2.1          988         494.0

spam read request            67        0        2.5         1161         580.5
spam read IO                  0        0        0.0            4           2.0

read stall count              1        0        0.1           51          25.5
read stall time               4        0        0.3          153          76.5

write IO                      8        0        1.7          812         406.0
write stall count             3        0        0.1           51          25.5
write stall time             24        0        0.5          223         111.5

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in this display, see the Performance Monitor help.

4.1.1.6 Performance Monitor Process Accounting Screen

OpenVMS VAX ≡ OpenVMS Alpha ≡

The Process Accounting screen provides continuously updated accounting information about local processes. This screen is an alternative to the OpenVMS SHOW PROCESS/CONTINUOUS utility. The screen provides equivalent information, except that all active database processes on a specific node can be monitored at the same time.

This screen shows direct operating system accounting information, thereby enabling a database administrator to evaluate the system resources used by database processes. The values in the Process Accounting screen are for all process activity, not just the activity that occurs while in the database. Therefore, this screen is useful for monitoring the complete application behavior.

You access the Process Accounting screen from the Process Information submenu.

This screen shows dynamically changing process information only. That is, quotas and other information that are fixed for each process are not displayed because that information can be obtained in other ways. The Process Accounting screen has brief and full modes that control the amount of information displayed for each active database process.

You select brief mode by typing B. In brief mode, one line per process is displayed, providing the following information: process ID, process name, CPU time, remaining lock quota count, page fault count, number of direct I/O operations, working set size, and virtual memory size. The following is an example of the Process Accounting screen in brief mode:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 10:22:20
Rate: 3.00 Seconds            Process Accounting           Elapsed: 00:20:11.30
Page: 1 of 1         SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1         Mode: Online
--------------------------------------------------------------------------------
Process.ID Process.name... CPUtime.... EnqCnt. PGflts. NumDio. WSsize. VMsize.
6E2012D3:1 RICK            00:00:15.46   17809   20775     808    4471   15082
6E201887:1 SMITH           00:00:11.23   17882   17383     444    3606   14426









--------------------------------------------------------------------------------
Exit Full Help Menu >next_page <prev_page Set_rate Write Zoom !
```

You select full mode by typing F. In full mode, a second line per process is displayed that provides the following additional information: user name, image name, process state, page file quota count, direct I/O quota count, number of buffered I/O operations, and buffered I/O quota count. The following is an example of the Process Accounting screen in full mode:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 10:24:48
Rate: 3.00 Seconds            Process Accounting           Elapsed: 00:22:39.16
Page: 1 of 1         SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1         Mode: Online
--------------------------------------------------------------------------------
Process.ID Process.name... CPUtime.... EnqCnt. PGflts. NumDio. WSsize. VMsize.
User.name... Image.name............... State.. PGfCnt. DioCnt. NumBio. BioCnt.
6E2012D3:1 RICK            00:00:15.55   17796   20776     808    4472   15082
RICK         SQL$601                   LEF      124448     200    1464     199
6E201887:1 SMITH           00:00:11.92   17858   18378     516    5000   14954
SMITH        SQL$601                   LEF      124576     200     515     199









--------------------------------------------------------------------------------
Brief Exit Help Menu >next_page <prev_page Set_rate Write Zoom !
```

Note that information on the Process Accounting screen cannot be reset. Because the information is gathered in real time, it cannot be written to the output file (the Output qualifier will not work with this screen) and, therefore, cannot be displayed during input file replay.

For information about each of the fields in the Process Accounting screen, see the Performance Monitor help. ♦

4.1.1.7 Performance Monitor Record Statistics Screen

The Record Statistics screen shows a summary of data row activity (rows marked, fetched, stored, or erased) for storage areas in the database. The following is an example of a Record Statistics screen:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 13:21:49
Rate: 3.00 Seconds             Record Statistics            Elapsed: 03:19:40.61
Page: 1 of 1     KODD$:[R_ANDERSON.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------

statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

record marked                 12       11        4.7         3784           2.1

record fetched                41       36       16.0        12730           7.1
    fragmented                 0        0        0.0            0           0.0

  record stored                6        5        2.3         1815           1.0
     fragmented                0        0        0.0            0           0.0
  pages checked                6        5        2.3         1815           1.0
       saved IO                0        0        0.0            0           0.0
      discarded                0        0        0.0            0           0.0

record erased                  0        0        0.0            0           0.0
   fragmented                  0        0        0.0            0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in this display, see the Performance Monitor help.

4.1.1.8 Performance Monitor AIJ Statistics Screen

The AIJ Statistics screen monitors both logical and physical after-image journaling activity. The following is an example of an AIJ Statistics screen:

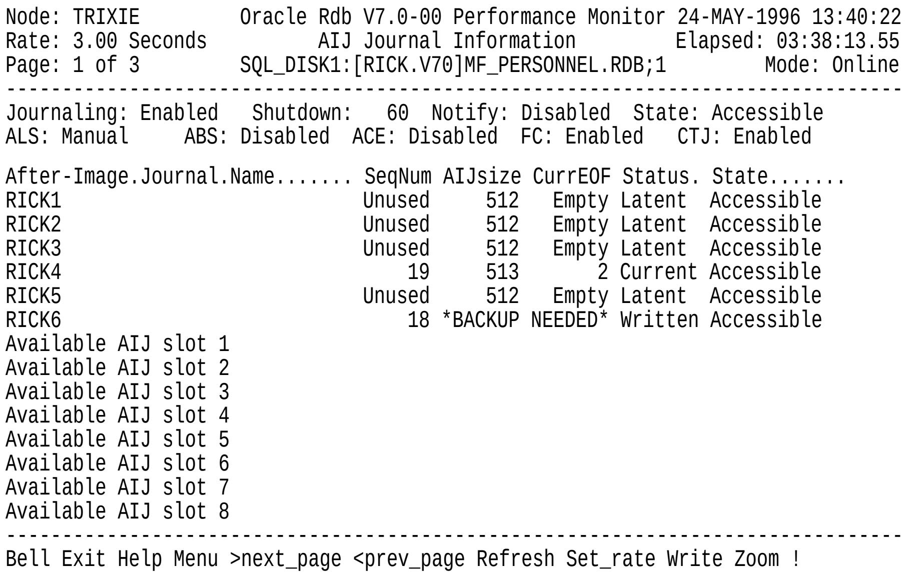
```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 13:29:50
Rate: 3.00 Seconds              AIJ Statistics            Elapsed: 03:27:41.92
Page: 1 of 1        SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------

statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

AIJ file writes               0        0       0.0            0           0.0
   data                       0        0       0.0            0           0.0
   control                    0        0       0.0            0           0.0
   file extend                0        0       0.0            0           0.0
   switch over                0        0       0.0            0           0.0

records written               0        0       0.0            1           0.3
blocks written                0        0       0.0            0           0.0
   filler bytes               0        0       0.0            0           0.0

lock rebuilds                 0        0       0.0            0           0.0
   AIJ file reads             0        0       0.0            1           0.3

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in this display, see the Performance Monitor help.

4.1.1.9 Performance Monitor AIJ Journal Information Screen

The AIJ Journal Information screen provides online information about all of a database's after-image journals on the current node. Most of the information displayed on the screen is obtained in real time, which means that the screen is automatically updated as AIJ database parameters are modified, or as AIJ operations such as a backup or journal switch-over are performed.

Because the AIJ Journal Information screen provides real-time information, the output is not recorded in the binary output file produced using the Output qualifier. Consequently, this screen is not available when you replay a binary file using the Input qualifier. You access the AIJ Journal Information screen from the Journaling Information submenu. The following is an example of the AIJ Journal Information screen:



```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 13:40:22
Rate: 3.00 Seconds         AIJ Journal Information        Elapsed: 03:38:13.55
Page: 1 of 3        SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
Journaling: Enabled   Shutdown:   60  Notify: Disabled  State: Accessible
ALS: Manual     ABS: Disabled  ACE: Disabled  FC: Enabled   CTJ: Enabled

After-Image.Journal.Name....... SeqNum AIJsize CurrEOF Status. State.......
RICK1                           Unused     512   Empty Latent  Accessible
RICK2                           Unused     512   Empty Latent  Accessible
RICK3                           Unused     512   Empty Latent  Accessible
RICK4                               19     513       2 Current Accessible
RICK5                           Unused     512   Empty Latent  Accessible
RICK6                               18 *BACKUP NEEDED* Written Accessible
Available AIJ slot 1
Available AIJ slot 2
Available AIJ slot 3
Available AIJ slot 4
Available AIJ slot 5
Available AIJ slot 6
Available AIJ slot 7
Available AIJ slot 8
--------------------------------------------------------------------------------
Bell Exit Help Menu >next_page <prev_page Refresh Set_rate Write Zoom !
```

The AIJ Journal Information screen contains information relating to AIJ journaling in general and information on each individual journal, including reserved AIJ journal slots.

Note that information on the AIJ Journal Information screen is for the current node only. Because an after-image journal is accessed by all nodes modifying the database, the information for one node could become stale. Therefore, the Refresh option on the horizontal menu at the bottom of the AIJ Journal Information screen is provided. The Refresh option causes the current node's information to be synchronized with all other nodes accessing the database.

For information about each of the fields shown in the AIJ Journal Information screen, see the Performance Monitor help.

See the *Oracle Rdb7 Guide to Database Maintenance* for a complete description of after-image journaling.

4.1.1.10 Performance Monitor Snapshot Statistics Screen

The Snapshot Statistics screen shows snapshot activity for both update and read-only transactions. The following is an example of a Snapshot Statistics screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 13:46:51
Rate: 3.00 Seconds           Snapshot Statistics          Elapsed: 03:44:42.12
Page: 1 of 1    RDBVMS_USER1:[LOGAN.MF_SAMPLE]MF_PERSONNEL.RDB;1    Mode: Online
--------------------------------------------------------------------------------

statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

Total transactions         1        0        0.1          6           1.0
R/O transactions           0        0        0.1          4           0.7

retrieved record           7        0        2.9        210          35.0
  fetched line             7        0        2.9        210          35.0
    read snap page         0        0        0.0          0           0.0

stored snap record         0        0        0.0          0           0.0
    page in use            0        0        0.0          0           0.0
    page too full          0        0        0.0          0           0.0
    page conflict          0        0        0.0          0           0.0
    extended file          0        0        0.0          0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in the Snapshot Statistics screen, see the Performance Monitor help.

4.1.1.11 Performance Monitor Checkpoint Statistics Screen

The Checkpoint Statistics screen shows transaction and checkpoint activity. Statistics are displayed for all processes on the node for a particular database. The total number of checkpoints, with a breakdown of the reasons for checkpointing, is displayed. The sum of all checkpoint intervals is also displayed, using three different metrics. You can use this information to compute the average interval between checkpoints, allowing you to decide if a checkpointing interval should be adjusted, and by how much.

Some of the columns provided by the Checkpoint Statistics screen measure events on a per second or per transaction basis. These columns are useful for measuring frequently occurring events such as I/O operations. Because checkpointing typically occurs at a slower rate, you will find the most meaningful checkpointing information in the total count column of the Checkpoint Statistics screen. Oracle Corporation recommends that you refer to this column when you use checkpoint statistics to determine optimal checkpoint intervals.

You access the Checkpoint Statistics screen from the Journaling Information submenu. The following is an example of a Checkpoint Statistics screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 24-MAY-1996 14:43:06
Rate: 3.00 Seconds          Checkpoint Statistics         Elapsed: 04:40:57.33
Page: 1 of 1        SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............ total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

transactions                  6        4       2.8          373           1.0
checkpoints                   1        0       0.1           16           0.0
    AIJ growth                1        0       0.1           16           0.0
    txn limit                 0        0       0.0            0           0.0
    time limit                0        0       0.0            0           0.0
    rollback                  0        0       0.0            0           0.0
    AIJ backup                0        0       0.0            0           0.0
    global                    0        0       0.0            0           0.0
interval: AIJ blks           28        0       3.2          423           1.1
interval: tx count           24        0       2.7          362           1.0
interval: seconds            11        0       0.6           87           0.2

checkpoint stall              0        0       0.0            0           0.0
flushed buffers               0        0       0.0            0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in the Checkpoint Statistics screen, see the Performance Monitor help.

Keep in mind that checkpointing influences recovery time. The main reason to consult checkpoint statistics is to find the average interval per checkpoint. You can use the information in the total count column to compute this average. For each category of checkpoint reason, use the average interval per checkpoint to help you decide if a checkpointing interval should be adjusted, and by how much.

You can set checkpoint limits for:

- .aij file growth

  You specify the .aij file growth limit by using the CHECKPOINT INTERVAL IS n BLOCKS clause of the ALTER DATABASE statement. If CHECKPOINT INTERVAL IS 1000 BLOCKS is specified, each process checkpoints when the .aij file has grown at least 1000 blocks since the process' last checkpoint.

- time

  You specify the time limit by using the CHECKPOINT TIMED EVERY n SECONDS clause of the ALTER DATABASE statement. If CHECKPOINT TIMED EVERY 600 SECONDS is specified, each process checkpoints when at least 10 minutes have elapsed since the process' last checkpoint.

- transactions

> You specify the transactions limit by using the RDM$BIND_CKPT_TRANS_INTERVAL logical name or the RDB_BIND_CKPT_TRANS_INTERVAL configuration parameter. If RDM$BIND_CKPT_TRANS_INTERVAL is defined as a system logical set to 10, each process checkpoints after 10 transactions unless a user redefines the RDM$BIND_CKPT_TRANS_INTERVAL logical name to a different value. That is, if a user defines RDM$BIND_CKPT_TRANS_INTERVAL as a process logical name and sets a value of 5, that user checkpoints after 5 transactions.

In the previous example, all of the checkpoints are triggered by the .aij file growth limit. This may mean that the transaction limit and time limit are set too high, or it may mean that .aij file growth limit is set too low. The rest of this section explains how to determine the average interval per checkpoint for each type of checkpoint limit. After you have determined the average interval per checkpoint for each type of checkpoint limit, you can reset the limits so that each type of checkpoint limit triggers approximately the same number of checkpoints, which results in optimal performance.

To find the average interval for a checkpoint, divide each of the interval categories by the total number of checkpoints. In the example, the seconds interval count is 87 and the total number of checkpoints is 16, so the average number of seconds between each time-triggered checkpoint is approximately 5.

Likewise, the transactions interval is 362 and the total number of checkpoints is 16, so the average number of transactions between checkpoints is approximately 23.

However, when computing the average interval in AIJ blocks, you must divide the AIJ block interval by the total number of checkpoints *minus the number caused by AIJ backups.* Although checkpoints caused by AIJ backups are counted in the total number of checkpoints, they are not counted in the total of AIJ block intervals. The following calculation shows that the .aij file grows by an average of approximately 26 blocks between each checkpoint.

$$Avg\ AIJ\ Blk\ Int = 423/(16 - 0) = 26.4375$$

Note that there may be more reasons for checkpoints than there are total checkpoints. This can occur because a single checkpoint may be triggered by more than one event. For example, a checkpoint may occur because of time and .aij file growth. Although both columns will increment by one, the total checkpoint column only increments by one.

4.1.1.12 Performance Monitor Checkpoint Information Screen

The Checkpoint Information screen displays process checkpoint information. One line of information is displayed for each process attached to the database on the current node. There may be blank lines in the display as processes detach from the database; in this way, once a process has attached to the database, it maintains the same location in the screen until it detaches from the database.

You access the Checkpoint Information screen from the Process Information submenu. The following is an example of a Checkpoint Information screen:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor  9-NOV-1995 10:08:36
Rate: 1.00 Second      Checkpoint Information (Unsorted)    Elapsed: 00:16:06.99
Page: 1 of 1      KODD_TEST:[R_ANDERSON.OE_MASTER]OE_RDB.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Process.ID  Ckpt.Vno:Ckpt.Vbn  QuietVno  Tx.Start.Time  AIJ: 2:8148
2083CE80:1s        2     7261
20819292:1s
20814D51:1         2     6251         2  10:07:29.76
20821752:1         2     6322         2  10:07:28.57
2082D156:1         2     7618         2  10:08:06.90
20822D57:1         2     6677         2  10:07:12.16
2081875B:1         2     7258         2  10:07:37.89

--------------------------------------------------------------------------------
Config Exit Help Menu >next_page <prev_page Refresh Set_rate Write Zoom !
```

For information on each of the fields shown in the Checkpoint Information screen, and the various configuration options, see the Performance Monitor help.

4.1.2 Managing Buffers

A **buffer** is an internal memory area used for temporary storage of database pages during read and update operations. When you request data from the database, Oracle Rdb reads into memory enough database pages to fill a buffer.

When you create or modify a database, you can set up buffers for database pages in either of two ways:

- You can choose user-private buffers (local buffers). This is the default.
- You can choose a common buffer pool for each node (global buffers).

With **local buffering**, Oracle Rdb maintains a local buffer pool for each user process. For more than one process to use the same page, each process must read the page from disk into its local buffer pool, even if the processes are running on the same node.

With **global buffering**, Oracle Rdb maintains one global buffer pool for each database. In a multinode system, each node maintains its own global buffer pool for each database in the multinode system, and the pools on different nodes are coordinated. A page in the global buffer pool can be used by more than one process at the same time, although only one process reads the page from disk into the global buffer pool.

In general, you should base your choice of local or global buffering on the extent to which data is shared in your database.

- If many processes frequently access the same pages, enabling global buffers can improve performance by reducing I/O and enhancing memory utilization.
- If a process repeatedly accesses the same group of pages, enabling global buffers can improve performance.
- If each process accesses its own small group of pages, enabling global buffers does not improve performance.

In this section and throughout this manual, the term user or process means a single attach to a database. Oracle Rdb considers multiple attaches from one process to be multiple users. This means that if a single user process or application attaches 10 times to a database, the database resources consumed will be 10 times what the single user process or application would consume in a single attach. In Example 4–1, user process SMITH with a process ID of 2080F11A attaches to the database once, and the RMU Show Users command shows that process 2080F11A is allocated 20 global buffers. User process RICK with a process ID of 20808D50 attaches to the database twice, and the RMU Show Users command shows that *each* attach is allocated the 20 buffers that are allocated to a single process. If database resources are consumed more quickly than you expect, use the RMU Show Users command to see whether one or more processes is attaching to the database multiple times.

Example 4–1 Each Database Attach by a Process Receives the Resources Allocated to a Single Process

```
SQL> -- User SMITH attaches to the database:
SQL> ATTACH 'FILENAME mf_personnel';
SQL> --
SQL> -- Show that the user process SMITH is allocated 20 global
SQL> -- buffers when attached to the database:
```

(continued on next page)

Example 4–1 (Cont.) Each Database Attach by a Process Receives the Resources Allocated to a Single Process

```
Oracle Rdb V7.0-00 on node TRIXIE 28-MAY-1996 09:22:18.51

database _$111$DUA176:[LOGAN.V70]MF_PERSONNEL.RDB;1
    - First opened 28-MAY-1996 09:21:53.86
    - current after-image journal file is
_$111$DUA176:[LOGAN.V70]RICK1.AIJ
;1
    - global buffer count is 1000
    - maximum global buffer count per user is 40
    - 980 global buffers free
    - 1 active database user
    - 2080F11A:1 - SMITH - non-utility, SMITH - active user
        - image $111$DUA600:[SQL_X07001.VAX.][CODE]SQL$701.EXE;1
        - 20 global buffers allocated
SQL> --
SQL> -- User process RICK from the same node attaches to the database
SQL> -- twice and each attach is allocated 20 global buffers:
SQL> ATTACH 'ALIAS MF_PERS1 FILENAME mf_personnel';
SQL> CONNECT TO 'ALIAS MF_PERS1 FILENAME mf_personnel' AS 'TEST';
SQL> $ RMU/SHOW USERS mf_personnel
Oracle Rdb V7.0-00 on node TRIXIE 28-MAY-1996 09:27:38.59

database _$111$DUA176:[LOGAN.V70]MF_PERSONNEL.RDB;1
    - First opened 28-MAY-1996 09:21:53.86
    - current after-image journal file is
_$111$DUA176:[LOGAN.V70]RICK1.AIJ
;1
    - global buffer count is 1000
    - maximum global buffer count per user is 40
    - 940 global buffers free
    - 3 active database users
    - 2080F11A:1 - SMITH - non-utility, SMITH - active user
        - image $111$DUA600:[SQL_X07001.VAX.][CODE]SQL$701.EXE;1
        - 20 global buffers allocated
    - 20808D50:1 - _RTA7: - non-utility, RICK - active user
        - image $111$DUA600:[SQL_X07001.VAX.][CODE]SQL$701.EXE;1
        - 20 global buffers allocated
    - 20808D50:2 - _RTA4: - non-utility, RICK - active user
        - image $111$DUA600:[SQL_X07001.VAX.][CODE]SQL$701.EXE;1
        - 20 global buffers allocated
SQL>
```

The Performance Monitor Buffer Information screen and the SQL SHOW DATABASE output in Example 4–2 show the Oracle Rdb local and global buffer parameters.

Example 4–2 Local and Global Buffer Parameters

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 21-JUN-1996 11:59:07
Rate: 3.00 Seconds            Buffer Information          Elapsed: 00:00:07.47
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------

Default user buffer count is 20  ❶
Default recovery buffer count is 20
Buffer size is 6 blocks  ❷
Global Buffers are disabled  ❸
- Global buffer count is 250  ❹
- Maximum global buffer count per user is 5  ❺
- Page transfer via memory is disabled
Global section size with global buffers disabled is 87600 bytes
- With global buffers enabled is 997430 bytes
Asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 5 buffers
Detected asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 4 buffers
- Pre-fetch threshold is 4 pages
Asynchronous batch-write is enabled
- Clean buffer count is 5
- Maximum batch size is 4 buffers
--------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
   .
   .
   .
$ SQL
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SHOW DATABASE *
Default alias:
    Oracle Rdb database in file MF_PERSONNEL
        Multischema mode is disabled
        Number of users:             50
        Number of nodes:             16
        Buffer Size (blocks/buffer):  6  ❷
        Number of Buffers:           20 ❶
        Number of Recovery Buffers:   20
        Snapshots are Enabled Immediate
        Carry over locks are enabled
        Lock timeout interval is 0 seconds
        Adjustable lock granularity is enabled
        Global buffers are disabled ❸ (number is 250, ❹ user limit is 5 ❺
   .
   .
   .
SQL>
```

The following callouts identify the buffer parameters in Example 4–2:

❶ The NUMBER OF BUFFERS parameter

The root file value for the NUMBER OF BUFFERS parameter specifies the default number of buffers to be allocated to each process that attaches to the database. Depending on whether local or global buffers are enabled and the setting of other buffer parameters, the number of buffers specified by this parameter may or may not be allocated to the process when it attaches to the database. See Section 4.1.2.2 for more information on the NUMBER OF BUFFERS parameter.

❷ The BUFFER SIZE parameter

This statement specifies the number of blocks per buffer. The value specified for the BUFFER SIZE parameter is valid for both local and global buffers. The value is stored in the database root file. See Section 4.1.2.1 for more information on the BUFFER SIZE parameter.

❸ The global buffers are disabled or enabled parameter

Local buffers are enabled by default (global buffers are disabled by default). When local buffers are enabled for a database, the values specified for the NUMBER IS parameter (callout ❹) and the USER LIMIT parameter (callout ❺) are not in effect for the database; these values are meaningful only when global buffers are enabled. See Section 4.1.2.6 for more information on enabling global buffers.

❹ The NUMBER IS parameter

The active value specified for the NUMBER IS parameter determines the total number of global buffers per node for a database. See Section 4.1.2.7 for more information on the NUMBER IS parameter.

❺ The USER LIMIT parameter

The active value specified for the USER LIMIT parameter determines the maximum number of global buffers that can be allocated to a process attached to the database.

In Example 4–2, the value for the USER LIMIT parameter in the mf_ personnel database is less than the value for the NUMBER OF BUFFERS parameter. Normally, when global buffers are enabled for a database, the USER LIMIT parameter should be increased to a value greater than the NUMBER OF BUFFERS parameter. See Section 4.1.2.8 for more information on the USER LIMIT parameter.

The parameter values displayed in Example 4–2 are the values stored in the database root file. The RMU Open command can be used to specify different values for the NUMBER IS and USER LIMIT parameters than those stored in the root file. See Section 4.1.2.7 and Section 4.1.2.8 for more information on using the RMU Open command to specify values for the NUMBER IS and USER LIMIT parameters, respectively.

4.1.2.1 Specifying Buffer Size

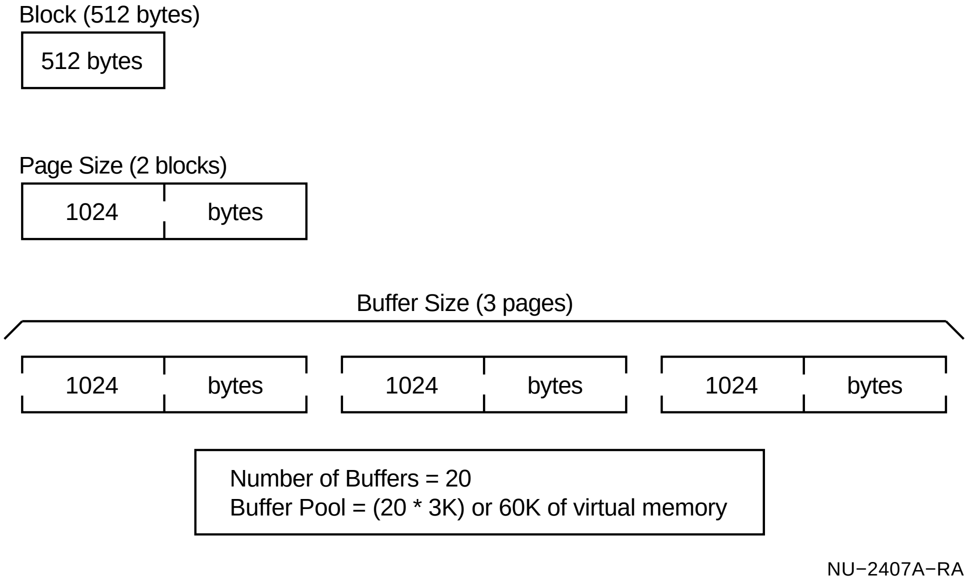
The value you assign to the BUFFER SIZE parameter of the SQL CREATE DATABASE or ALTER DATABASE statement determines the number of database pages each buffer can contain. This value is valid for both local and global buffering. The default is three times the value of the page size. If the default page size is used, the default buffer size is 6 blocks (three times the default page size of 2 blocks), or 3072 bytes. Figure 4–1 shows the relationships among BLOCK SIZE, PAGE SIZE, and BUFFER SIZE defaults.

The buffer size you select can affect performance and depends upon the general type of database operations. If you specify a large buffer size, it may reduce I/O operations for sequential retrievals because the system reads a buffer's worth of contiguous pages, and provides some read-ahead capability.

Random data retrieval, however, can benefit from a smaller buffer size. When the required page is not in a buffer, Oracle Rdb must read in a buffer's worth of pages from disk. The larger the buffer size, the more blocks Oracle Rdb must read to fill one buffer.

Buffer size is a database-wide parameter. The number of blocks for each page and buffer is restricted to less than 64 blocks. However, page size can vary by storage area for multifile databases, and the page size should be determined by the sizes of the records that will be stored in each storage area. You can change the buffer size after a database is created by using the SQL EXPORT and IMPORT statements or by using the BUFFER SIZE IS clause of the ALTER DATABASE statement. Refer to the *Oracle Rdb7 SQL Reference Manual* for more information on these SQL statements.

Figure 4–1 Buffer Pool: Database Parameter Defaults



When you assign a value to the BUFFER SIZE parameter, choose a number that is evenly divisible by all page sizes for all storage areas in your multifile database. This avoids wasted memory. For example, if you have three storage areas with page sizes of 16, 24, and 32 blocks respectively, choose a buffer size of 96 blocks to ensure optimal buffer utilization. Choosing a buffer size of 64 blocks would be inefficient because, for the storage area with a page size of 24 blocks, you waste 16 blocks (25 percent) of each buffer. Oracle Rdb reads as many pages as can fit into the buffer, and in this instance, it reads two 24-block pages into the buffer, leaving 16 wasted blocks.

If you define or modify a page size and specify a size larger than the buffer size, Oracle Rdb returns the following error message:

```
RDMS-F-BUFSMLPAG, buffer size is less than page size.
```

4.1.2.2 Specifying the Default Number of User Buffers

The value you specify for the NUMBER OF BUFFERS parameter determines the default number of buffers Oracle Rdb allocates to each user process that attaches to the database. Depending on whether local or global buffers are enabled and the setting of other buffer parameters (including the number of buffers specified by the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter), the number of buffers specified by the NUMBER OF BUFFERS parameter may or may not be allocated to the process when it attaches to the database.

Note

Do not confuse the NUMBER OF BUFFERS parameter with the global buffer parameter NUMBER IS. The NUMBER IS parameter specifies the total number of buffers in the global buffer pool and has meaning only within the context of global buffering. Refer to Section 4.1.2.6 for information on enabling global buffers.

You can specify the NUMBER OF BUFFERS parameter for a database by using the NUMBER OF BUFFERS clause of the SQL ALTER DATABASE and CREATE DATABASE statements:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> NUMBER OF BUFFERS IS m;
```

You can also specify the NUMBER OF BUFFERS parameter for a database by using the Local_Buffers=(Number=m) qualifier of the RMU Restore command:

```
$ RMU/RESTORE/LOCAL_BUFFERS=(NUMBER=m)/NOCDD mf_pers_backup
$ rmu -restore -local_buffers=\(number=m\) mf_pers_backup
```

When you use the ALTER DATABASE and CREATE DATABASE statements or the RMU Restore command to specify a value for the NUMBER OF BUFFERS parameter, the value is stored in the database root file, regardless of whether the database has local buffers or global buffers enabled.

You can also use the logical name RDM$BIND_BUFFERS or the configuration parameter RDB_BIND_BUFFERS to specify the number of buffers a process can be allocated when it attaches to a database (that is, to specify a different value than the value specified for the NUMBER OF BUFFERS parameter). A positive integer value must be assigned with the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter for it to have any effect. For example, the process issuing the following command is requesting that 35 buffers be allocated to it when it attaches to a database:

```
$ DEFINE RDM$BIND_BUFFERS 35
```

The following text describes how Oracle Rdb determines the number of buffers a user process is allocated when it attaches to a database, based on the local and global buffer parameters in effect for the database:

- When local buffers are enabled for a database, Oracle Rdb allocates each process that attaches to the database the number of buffers specified by the NUMBER OF BUFFERS parameter (stored in the root file) unless the process specifies a different value with the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter. In that case, Oracle Rdb allocates the process the number of buffers specified with the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter.
- When global buffers are enabled for a database, Oracle Rdb determines how many buffers to allocate to a process based on the values of the following parameters:
  - The USER LIMIT parameter

    The active USER LIMIT parameter for a database determines the maximum number of global buffers that Oracle Rdb can allocate to a process using the database. There are two ways to set the value for the USER LIMIT parameter for a database. One way is to set the value using either the SQL CREATE DATABASE or ALTER DATABASE statements or the RMU Restore Global_Buffers=(User_Limit=t) command, which stores the value in the database root file. The other way is to use the RMU Open Global_Buffers=(User_Limit=t) command, which opens a database and specifies the maximum number of buffers that processes can be allocated while the database is open. The value specified for the USER LIMIT parameter with the RMU Open Global_Buffers command overrides the USER LIMIT value stored in the database root file until the database is closed. Use the RMU Show Users database-name command to display the active value for the USER LIMIT parameter (the value currently in effect for the database on the current node).

    See Section 4.1.2.8 for more information and examples on setting the USER LIMIT parameter.
  - The RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter

  When the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter is defined for a process, Oracle Rdb allocates that process the number of buffers specified by RDM$BIND_BUFFERS or RDB_BIND_BUFFERS if this number specifies fewer buffers than the active USER LIMIT value. If RDM$BIND_BUFFERS or RDB_BIND_BUFFERS specifies more buffers than the active USER LIMIT value, the process receives the number of buffers specified by the active USER LIMIT value.

- The NUMBER OF BUFFERS parameter

  When a process does not define the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter, Oracle Rdb allocates the process the number of buffers specified by the NUMBER OF BUFFERS parameter if the NUMBER OF BUFFERS value specifies fewer buffers than the active USER LIMIT value. If the NUMBER OF BUFFERS parameter specifies more buffers than the active USER LIMIT value, the process receives the number of buffers specified by the active USER LIMIT value.

You can use the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter to tailor the buffer requirements of individual processes. For example, a large batch job running at night may need more buffers than interactive processes running during the day. By defining RDM$BIND_BUFFERS or RDB_BIND_BUFFERS for the batch job process, you can grant extra buffers to the batch job (unless a database has global buffers enabled and the active value for the USER LIMIT parameter for the database is smaller than the value specified by the logical name RDM$BIND_BUFFERS or the configuration parameter RDB_BIND_BUFFERS) and maintain the default number of buffers for other processes that access the database. Note that, by default, the value specified by the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter is specific to the defining process and applies to all databases used by the process.

OpenVMS VAX OpenVMS Alpha

If you have the required privileges, you can define RDM$BIND_BUFFERS as a group or system logical name to affect a larger group of processes. ♦

One of the greatest benefits of increasing the number of buffers is that more index nodes stay in the buffer pool. Consequently, I/O operations may be reduced. Also, response time can improve because more output is deferred until commit time, which uses asynchronous I/O operation. This is especially beneficial for a multiuser database.

If you do not specify a value for the NUMBER OF BUFFERS parameter, the default value is 20 buffers. When a user process requests data not contained in the current buffer pool, Oracle Rdb reuses the buffers, writing updated buffers to the database to make room for new pages.

4.1.2.3 Tuning Local Buffers

The values you supply for the database parameters BUFFER SIZE and NUMBER OF BUFFERS determine the amount of virtual memory reserved for the buffer pool for each database user when local buffers are enabled. Using the default parameter values (shown in Figure 4–1), yields the following calculation:

$$Buffer\ Pool = Number\ of\ Buffers * Buffer\ Size$$

$$Buffer\ Pool = 20 * (6 * 512) = 61440\ Bytes$$

Thus, by default, each user has a buffer pool of 61,440 bytes or 120 blocks. Along with the actual memory consumed by the buffers, some overhead is associated with buffer management, so the actual size of the buffer pool needs to be a little more than 120 blocks.

In some rare cases, if the buffer pool is too large relative to available physical memory and quotas, the operating system may be forced to perform virtual paging of the buffer pool in addition to reading a database page. This may degrade performance. To avoid paging, you should make sure that the sum memory requirements for all users active at the same time on a node is smaller than the amount of physical memory.

If the buffer pool is too small, Oracle Rdb may have to perform more I/O operations to bring the database pages into memory.

You can determine buffer size and the number of buffers required for your database by analyzing row sizes in the database and the types of database operations users perform.

- If you specify a high BUFFER SIZE value, Oracle Rdb can read in more data, or find related rows when they are stored near each other, with each I/O operation. This works well for sequential data retrieval.
- If you specify a NUMBER OF BUFFERS parameter greater than the default (20), Oracle Rdb is more likely to retain previously used rows in the user's buffer pool. This works well for applications that refer to the same data multiple times. Indexes are a good example of applications that refer to the same data multiple times. If you use sorted indexes for range retrievals, you may want to specify a larger number of buffers. You may also want to enable global buffers if your application refers to the same data multiple times.

Generally, these defaults are adequate for workloads with mixed transaction types. However, if you have a transaction type that occurs more frequently than others, you may want to adjust the values of these database parameters. Having many small buffers is generally better than having a few large buffers because it reduces I/O operations and avoids virtual paging of the buffer pool. Refer to Section 8.1.3 for more information on local buffers and memory consumption.

If queries retrieve rows based on indexed columns, Oracle Rdb uses the database key stored in the index to find the rows directly. However, Oracle Rdb reads the entire page that contains the row as well as enough database pages to fill a buffer in the user's buffer pool. Oracle Rdb always reads a full buffer of data from the database to the user's buffer pool.

During sequential retrieval of data in the database, Oracle Rdb reads successive pages from the database file to the user's buffer pool. The buffer size and number of buffers, therefore, determine how much data is read and made immediately available to the user without further I/O operation. The next row required by the user, and not already in the buffer pool, requires Oracle Rdb to flush those user buffers that contain data that has not been used recently. If the user has modified any rows, the buffers that contain those rows must first be written back to the database file before they can be used again. If the data has not been modified, the buffers can be reused immediately.

Use the Performance Monitor's PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches screens to see the effects of the number of buffers and to determine what adjustment you may need to make to improve performance (see Example 8–8 and Example 8–9).

4.1.2.4 Locking Local Buffers into Physical Memory

OpenVMS Alpha

Oracle Rdb includes a performance enhancement for databases accessed from OpenVMS Alpha systems. This feature uses the OpenVMS buffer object feature to lock Oracle Rdb local buffers into physical memory. Locking Oracle Rdb local buffers into memory increases symmetric multiprocessing (SMP) parallelism and scaling and improves I/O performance by eliminating OpenVMS overhead.

To utilize this feature you must:

- Run Oracle Rdb along with OpenVMS Alpha Version 6.1 or higher on an Alpha uniprocessor or an SMP system.
- Define the logical name RDM$BIND_BUFOBJ_ENABLED to be any value.

- Increase the OpenVMS user quota BYTLM by the number of bytes in your local buffer pool. For example, if the number of local buffers is 1000, and the buffer size is 8 blocks, determine the value of the BYTLM parameter using the following formula:

$$1000 * (8 * 512) = 4,096,000\ bytes$$

Do not use this feature on systems that are memory constrained. OpenVMS pages that are defined as a buffer object can be paged and swapped, but must remain resident in physical memory. This physical memory becomes unavailable for use by any other process until the image is terminated.

Note

The Oracle Rdb utilization of OpenVMS buffer objects is not available on databases with global buffers enabled.

♦

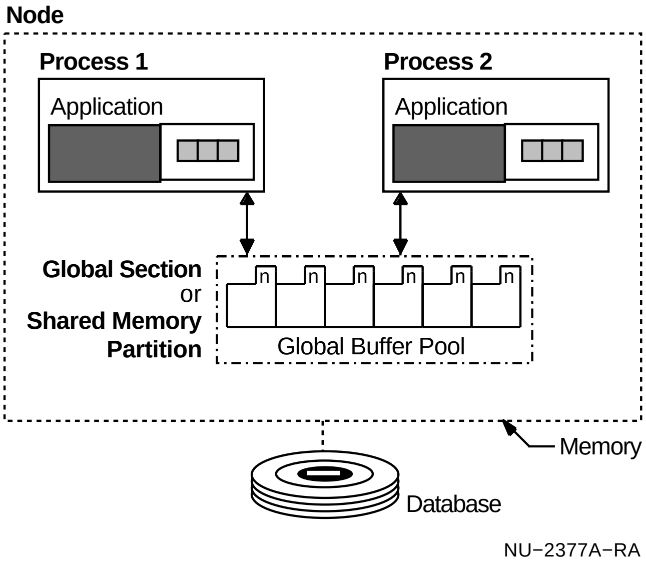
4.1.2.5 Global Buffer Pools

When a database has local buffers enabled, Oracle Rdb manages a separate buffer pool for each user *process*. When a database has global buffers enabled, Oracle Rdb manages a global buffer pool for the database on each *node* from which user processes are accessing the database. All the users accessing a database from a node use the global buffer pool for the database on that node. So, when global buffers are enabled for a database, the global buffer pool for the database is comprised of the global buffer pools for the database from the individual nodes from which users are accessing the database.

You can enable global buffering with the GLOBAL BUFFERS ARE ENABLED qualifier to the SQL ALTER or CREATE DATABASE statements, or with the RMU Restore Global_Buffers=Enabled command. See Section 4.1.2.6 for more information on enabling global buffers.

Figure 4–2 shows an example of global buffer management for two processes. In this discussion, a user or process is considered a single attach. Oracle Rdb considers multiple attaches from one process to be multiple users. This means that if a single user process attaches 10 times to a database, the database resources consumed will be 10 times what the single user process would consume in a single attach. Each attach makes calls to Oracle Rdb for database pages. In turn, Oracle Rdb calls the operating system to retrieve the pages from the storage devices into the global buffer pool.

Figure 4–2 Global Buffer Management



Oracle Rdb sets up a global buffer pool (an in-memory cache of buffers) in a global section on OpenVMS and in a shared memory partition on Digital UNIX to receive buffers of pages. User processes then map the global section or shared memory partition to their virtual memory. This increases the virtual memory requirements for users (compared to using local buffers), but physical memory is better utilized. Only one copy of each page resides in the global buffer pool but each attached process running on the node can simultaneously read the same page. When global buffers are enabled for a database, the system may require additional global section or shared memory partition resources. You can also use the Performance Monitor Buffer Information screen to estimate the size of the global section or shared memory partition before you enable global buffers for the database, as explained in Section 4.1.2.12.

While each process can access any buffer in the pool, the number of buffers a process can *use* at a given time is restricted and referred to as an allocate set. The number of buffers in a user's allocate set is determined by which of the following values is in effect when the user attaches to the database:

- The NUMBER OF BUFFERS value
- The RDM$BIND_BUFFERS or RDB_BIND_BUFFERS value

- The USER LIMIT IS value

Section 4.1.2.2 describes how Oracle Rdb determines the number of buffers a user process is allocated when it attaches to a database, based on the local and global buffer parameters in effect for the database.

If the global buffer pool is full and a process requests a page not contained in the pool, Oracle Rdb must first free an existing global buffer. To free a global buffer, Oracle Rdb uses a modified version of the least-recently used (LRU) replacement policy and reads new pages into the emptied global buffer.

The remainder of this section uses a simple example to illustrate how global buffering works.

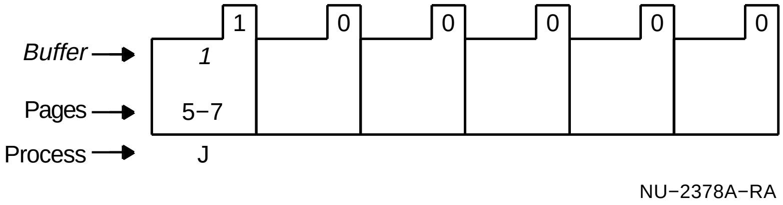
The example makes the following assumptions:

- Two processes are attached to the database; each process has an allocate set of 3 buffers.
- All the pages are read from a uniform format storage area. The first page in the storage area is a SPAM page. SPAM pages and data pages are not read together (that is, Oracle Rdb does not read SPAM and data pages in the same buffer).
- The buffer size is 6 blocks and page size is 2 blocks, so the size of each clump is 3 pages. A clump of pages that is read will always be the same pages, regardless of which page in the clump is requested. If a user requests page 2 or 3 or 4, the same three pages (2–4) will be read.
- Global buffering is enabled; 6 buffers are in the global buffer pool.

Therefore, the global buffer pool contains a total of 36 disk blocks (18 database pages) for the database. All 18 pages in the buffer pool are available to each process, but a process can look at only 9 pages (the 3 buffers in its allocate set) at any given time.

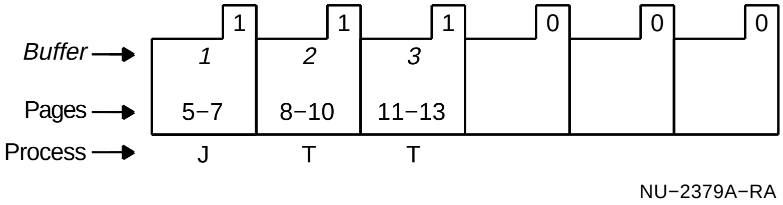
Figures 4–3 through 4–8 show the pages read into global buffers by two processes, J and T. Process J executes a query that causes three pages to be read into the first global buffer. Because a page can be referenced by multiple processes, Oracle Rdb associates a counter with each buffer that notes how many processes are referencing the buffer. In Figure 4–3, the counter is set to 1, which indicates a single process is accessing the buffer.

Figure 4–3 Global Buffer Pool: Buffer 1



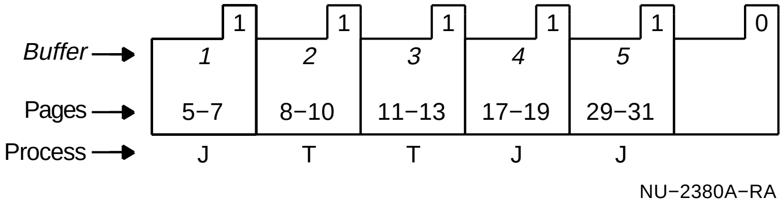
Process T now begins processing, and Oracle Rdb reads six more pages into the global buffer pool, which fills two buffers, as shown in Figure 4–4.

Figure 4–4 Global Buffer Pool: Buffers 1–3



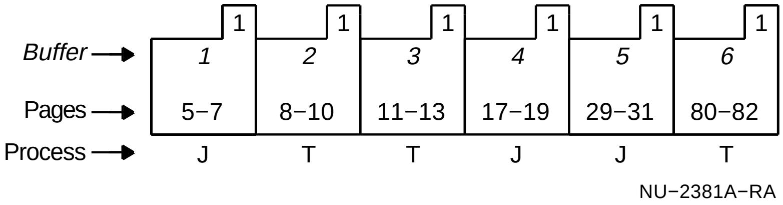
Process J now asks Oracle Rdb to read six more pages into the global buffer pool, as shown in Figure 4–5.

Figure 4–5 Global Buffer Pool: Buffers 1–5



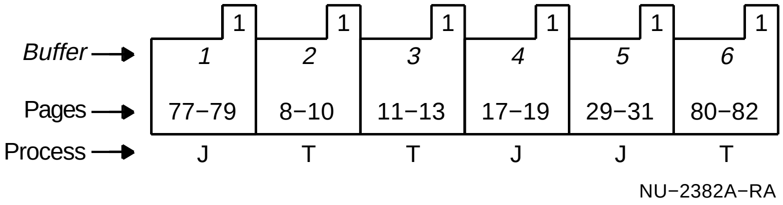
A subsequent call by process T reads in three more pages, thus filling up the remaining global buffer in the global buffer pool, as shown in Figure 4–6.

Figure 4–6 Global Buffer Pool: Buffers 1–6



Process J now wants to read page 77, which is not in the buffer pool. Because process J has reached its maximum allotment of three buffers, it must give up a global buffer before page 77 can be read into the buffer pool. Applying the LRU policy to the buffers in process J's allocate set, the first global buffer is flushed and filled with three more pages, one of which is the required page, as shown in Figure 4–7.

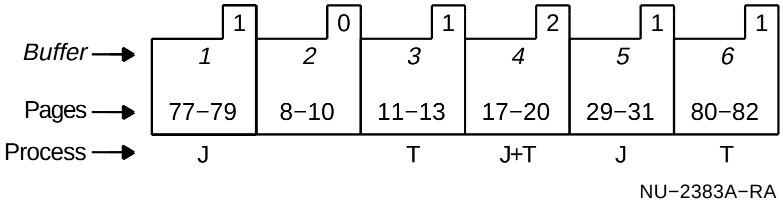
Figure 4–7 Global Buffer 1 Changes



Process T now wants to read page 17. Because process T has reached its maximum allotment of three buffers, process T must first give up a global buffer. Applying the LRU policy to process T's allocate set, process T releases global buffer number 2, thus setting the counter to 0. Although no processes are reading pages 8–10, the pages remain in the global buffer (refer to Figure 4–8). If process T had modified a page in the second global buffer, the modified page would have been written to disk before the buffer was de-allocated.

Because process J read pages 17–19 into global buffer 4, process T can read page 17 without performing any I/O. The counter for global buffer 4 now reflects two users accessing this range of pages. See Figure 4–8.

Figure 4–8 Global Buffers 2 and 4 Change



Any process can read pages 8–10 without executing an I/O. However, if a process requires pages not currently in the global buffer pool, pages 8–10 are replaced with new pages.

4.1.2.6 Enabling Global Buffers

When global buffers are enabled for a database in a multinode system, each node has a global buffer pool for each database. You can enable global buffers for a database by using the GLOBAL BUFFERS ARE ENABLED clause of the SQL ALTER DATABASE and CREATE DATABASE statements:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> GLOBAL BUFFERS ARE ENABLED;
```

There must not be any users attached to the database when you enable global buffering.

You can disable global buffers for a database by using the GLOBAL BUFFERS ARE DISABLED clause of the SQL ALTER DATABASE and CREATE DATABASE statements:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> GLOBAL BUFFERS ARE DISABLED;
```

There must not be any users attached to the database when you disable global buffering.

You can also enable global buffers for a database by using the Global_Buffers=Enabled qualifier of the RMU Restore command:

```
$ RMU/RESTORE/GLOBAL_BUFFERS=ENABLED/NOCDD mf_pers_backup
$ rmu -restore -global_buffers=enabled mf_pers_backup
```

You can disable global buffers for a database by using the Global_Buffers=Disabled qualifier of the RMU Restore command:

```
$ RMU/RESTORE/GLOBAL_BUFFERS=DISABLED/NOCDD mf_pers_backup
$ rmu -restore -global_buffers=disabled mf_pers_backup
```

When you use the ALTER DATABASE and CREATE DATABASE statements or the RMU Restore command to enable or disable global buffers for a database, the setting is stored in the database root file.

When a database has local buffers enabled, the values specified for the NUMBER IS and USER LIMIT parameters are not in effect for the database; the values for these global buffer parameters are meaningful only when global buffers are enabled. Other buffer parameter values, such as NUMBER OF BUFFERS and BUFFER SIZE, are valid for *both* local and global buffers.

OpenVMS VAX OpenVMS Alpha Because the OpenVMS operating system does not allow a global section size change, more buffers cannot be added to the global buffer pool as more users attach to the database. It is important, therefore, to select the right combination of values for the default buffer count per user (the NUMBER OF BUFFERS parameter), the number of global buffers (the NUMBER IS parameter), and the maximum number of buffers per user (the USER LIMIT parameter). ♦

Global buffers can require additional global section or shared memory partition resources. For more information, see Section 4.1.2.12.

4.1.2.7 NUMBER IS Parameter

The NUMBER IS parameter specifies the total number of global buffers per node for a database. You can specify the NUMBER IS parameter for a database by using the NUMBER IS clause of the SQL ALTER DATABASE and CREATE DATABASE statements:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>    GLOBAL BUFFERS ARE ENABLED
cont>    (NUMBER IS n);
```

You can also specify the NUMBER IS parameter for a database by using the Global_Buffers=Total=n qualifier of the RMU Restore command:

```
$ RMU/RESTORE/GLOBAL_BUFFERS=(TOTAL=n)/NOCDD mf_pers_backup
$ rmu -restore -global_buffers=\(total=n\) mf_pers_backup
```

When you use the ALTER DATABASE and CREATE DATABASE statements or the RMU Restore command to specify a value for the NUMBER IS parameter, the value is stored in the database root file. The value stored with these statements or commands should be appropriate for the node in the cluster with the least physical memory.

You can also specify the NUMBER IS parameter for a database by using the Global_Buffers=Total=n qualifier of the RMU Open command:

```
$ RMU/OPEN/GLOBAL_BUFFERS=(TOTAL=n) mf_personnel
$ rmu -open -global_buffers=\(total=n\) mf_personnel
```

The RMU Open Global_Buffers=(Total=n) command specifies the total number of global buffers for the database on the node from which the command is issued. You might want to increase the number of global buffers for a node with a large amount of physical memory when it is clustered with a node that has less physical memory. By having more global buffers for the larger memory, you can improve performance on that node. On the node from which the RMU Open Global_Buffers=(Total=n) command is issued, the global buffer pool contains the number of global buffers specified by n while the database is open.

One benefit of using the RMU Open Global_Buffers=(Total=n) command is that Oracle Rdb does not have to remap the buffer pool every time the database is opened. Refer to the *Oracle RMU Reference Manual* for more information on the RMU Open command.

The Performance Monitor Buffer Information screen and the SQL SHOW DATABASE output display the global buffer parameters stored in a database root file, and the RMU Show Users command displays the active values for the global buffer parameters (the active parameters may be different than the root file if the RMU Open Global_Buffers command was used to open the database on the node). In Example 4–3, the Buffer Information screen shows that the value for the NUMBER IS parameter in the root file is 250 global buffers, while the RMU Show Users command in Example 4–4 shows that the active value for the NUMBER IS parameter for the database on the current node is 350 global buffers.

Example 4–3 Using the Buffer Information Screen to Display the Number of Global Buffers

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 21-JUN-1996 11:59:07
Rate: 3.00 Seconds            Buffer Information           Elapsed: 00:00:07.47
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
---------------------------------------------------------------------------------

Default user buffer count is 20
Default recovery buffer count is 20
Buffer size is 6 blocks
Global Buffers are disabled
- Global buffer count is 250              <------------ Notice
- Maximum global buffer count per user is 5
- Page transfer via memory is disabled
Global section size with global buffers disabled is 87600 bytes
- With global buffers enabled is 997430 bytes
Asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 5 buffers
Detected asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 4 buffers
- Pre-fetch threshold is 4 pages
Asynchronous batch-write is enabled
- Clean buffer count is 5
- Maximum batch size is 4 buffers
---------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
```

Example 4–4 Using the RMU Show Users Command to Display the Active Value for the NUMBER IS Parameter

```
$ RMU/OPEN/GLOBAL_BUFFERS=(TOTAL=350) mf_personnel
$ RMU/SHOW USERS mf_personnel
Oracle Rdb V7.0-00 on node TRIXIE 28-MAY-1996 10:08:21.49

database _$111$DUA176:[LOGAN.V70]MF_PERSONNEL.RDB;1
    - First opened 28-MAY-1996 10:02:47.44
    * database is opened by an operator
    - current after-image journal file is
_$111$DUA176:[LOGAN.V70]RICK1.AIJ;1
    - global buffer count is 350          <------------ Notice
    - maximum global buffer count per user is 25
    - 325 global buffers free
    - 1 active database user
    - 2080932F:1 - RICK - non-utility, RICK - active user
        - image $111$DUA600:[SQL_X07001.VAX.][CODE]SQL$701.EXE;1
        - 25 global buffers allocated
$
```

The default for the NUMBER IS parameter is five times the maximum number of users. You should base this value on the number of users, the needs of the database, and available physical memory. Section 4.1.2.9 suggests a method for determining the number of global buffers.

Do not define more global buffers than there is memory available for on your system or the system will experience page faults. In a multinode system, the memory capacity of the smallest node is the limiting factor, although the NUMBER IS and USER LIMIT parameters can be tailored to the resources available on different nodes of a multinode system by using the RMU Open command.

The sum of the buffers allocated to all users cannot be greater than the total number of global buffers specified by the NUMBER IS parameter. This ensures that free buffers are available when a user needs to read a new buffer into the buffer pool. If the combined user buffer demand exceeds the size of the global buffer pool (resulting in attaches being denied to the database), increase the value of the NUMBER IS parameter. If the demand exceeds the size of physical memory, you need more memory on your system or you need to distribute users among different nodes of a multinode system. In this situation, you can also allow page faulting by defining NUMBER IS to be more than the capacity of the physical memory. Page faulting degrades performance, but this situation is permitted.

4.1.2.8 USER LIMIT Parameter

The USER LIMIT parameter specifies the maximum number of global buffers that can be allocated to a process. If global buffers are disabled for a database, the value specified for the USER LIMIT parameter is meaningless because global buffers are not used.

When you enable global buffers for a database, you will usually want the value for the USER LIMIT parameter to be greater than the value for the NUMBER OF BUFFERS parameter. If the USER LIMIT value is less than the NUMBER OF BUFFERS value, users will receive fewer buffers with global buffers enabled than they did when the database had local buffers enabled.

You can set a value for the USER LIMIT parameter by specifying the USER LIMIT IS clause of the SQL ALTER DATABASE and CREATE DATABASE statements:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>    GLOBAL BUFFERS ARE ENABLED
cont>    (USER LIMIT IS t);
```

You can also specify the USER LIMIT parameter for a database by using the Global_Buffers=(User_Limit=t) qualifier of the RMU Restore command:

```
$ RMU/RESTORE/GLOBAL_BUFFERS=(USER_LIMIT=t)/NOCDD mf_pers_backup
$ rmu -restore -global_buffers=\(user_limit=t\)  mf_pers_backup
```

When you use the ALTER DATABASE and CREATE DATABASE statements or the RMU Restore command to specify a value for the USER LIMIT parameter, the value is stored in the database root file. The value you store with these statements or commands should be appropriate for the node in the cluster with the least physical memory.

You can also specify the USER LIMIT parameter for a database by using the Global_Buffers=(User_Limit=t) qualifier of the RMU Open command:

```
$ RMU/OPEN/GLOBAL_BUFFERS=(USER_LIMIT=t) mf_personnel
$ rmu -open -global_buffers=\(user_limit=t\) mf_personnel
```

The RMU Open Global_Buffers=(User_Limit=t) command specifies the maximum number of global buffers that can be allocated to a process attached to the database on the node from which the command is issued. You might want to increase the number of global buffers per process for a node with a large amount of physical memory when it is clustered with a node that has less physical memory. By having more global buffers per process for the larger memory, you can improve performance on that node. On the node from which the RMU Open Global_Buffers=(User_Limit=t) command is issued, the maximum number of global buffers that can be allocated to a process while the database is open is the value specified by t (the USER LIMIT value specified with the RMU Open Global_Buffers command overrides the USER LIMIT value stored in the database root file until the database is closed. The RMU Open Global_Buffers=(User_Limit=t) command fails if the database is already open. Refer to the *Oracle RMU Reference Manual* for more information on the RMU Open command.

The Buffer Information screen and the SQL SHOW DATABASE output display the global buffer parameters stored in a database root file, and the RMU Show Users command displays the active values for the global buffer parameters (the active parameters may be different than the root file if the RMU Open Global_Buffers command was used to open the database on the node). In Example 4–5, the Buffer Information screen shows that the value for the USER LIMIT parameter in the root file is 25 global buffers, while the RMU Show Users command in Example 4–6 shows that the active value for the

USER LIMIT parameter for the database on the current node is 50 global buffers.

Example 4–5 Using the Performance Monitor Buffer Information Screen to Display the Active Value for the USER LIMIT Parameter

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 21-JUN-1996 11:59:07
Rate: 3.00 Seconds            Buffer Information          Elapsed: 00:00:07.47
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------

Default user buffer count is 20
Default recovery buffer count is 20
Buffer size is 6 blocks
Global Buffers are disabled
- Global buffer count is 250
- Maximum global buffer count per user is 25  <------------ Notice
- Page transfer via memory is disabled
Global section size with global buffers disabled is 87600 bytes
- With global buffers enabled is 997430 bytes
Asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 5 buffers
Detected asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 4 buffers
- Pre-fetch threshold is 4 pages
Asynchronous batch-write is enabled
- Clean buffer count is 5
- Maximum batch size is 4 buffers
--------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
```

Example 4–6 Using the RMU Show Users Command to Display the Active Value for the USER LIMIT Parameter

```
$ RMU/OPEN/GLOBAL_BUFFERS=(USER_LIMIT=50) mf_personnel
Oracle Rdb V7.0-00 on node TRIXIE 28-MAY-1996 10:21:08.13

database _$111$DUA176:[LOGAN.V70]MF_PERSONNEL.RDB;1
    - First opened 28-MAY-1996 10:20:53.50
    * database is opened by an operator
    - current after-image journal file is
_$111$DUA176:[LOGAN.V70]RICK1.AIJ;1
    - global buffer count is 250
    - maximum global buffer count per user is 50  <------------ Notice
    - 200 global buffers free
    - 1 active database user
    - 2080932F:1 - RICK - non-utility, RICK - active user
        - image $111$DUA600:[SQL_X07001.VAX.][CODE]SQL$701.EXE;1
        - 50 global buffers allocated
```

When global buffers are enabled for a database, a process can define a value with the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter that specifies the number of global buffers to be allocated to the process when it attaches to a database. You must assign a positive integer value with RDM$BIND_BUFFERS or RDB_BIND_BUFFERS for it to have any effect. The process receives the number of global buffers specified by RDM$BIND_BUFFERS or RDB_BIND_BUFFERS when that value is lower than the active value for the USER LIMIT parameter for the database, as shown in Example 4–7.

Example 4–7 Using the RDM$BIND_BUFFERS Logical Name to Specify Fewer Global Buffers Than the Value Specified by the Active USER LIMIT Parameter

```
$ ! A user executes the following commands:
$ DEFINE RDM$BIND_BUFFERS 8
$ !
$ SQL
SQL> ATTACH 'FILENAME mf_personnel';
   .
   .
   .
$!
$! Another user on the same node as the first user executes
$! the following command:
$ RMU/SHOW USERS mf_personnel
Oracle Rdb V7.0-00 on node TRIXIE 28-MAY-1996 10:27:59.10

database _$111$DUA176:[LOGAN.V70]MF_PERSONNEL.RDB;1
    - First opened 28-MAY-1996 10:20:53.50
    * database is opened by an operator
    - current after-image journal file is
_$111$DUA176:[LOGAN.V70]RICK1.AIJ;1
    - global buffer count is 250
    - maximum global buffer count per user is 25
    - 242 global buffers free
    - 1 active database user
    - 2080F11A:1 - RICK - non-utility, RICK - active user
        - image $111$DUA600:[SQL_X07001.VAX.][CODE]SQL$701.EXE;1
        - 8 global buffers allocated
$
```

If the value specified by the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter is higher than the active value for the USER LIMIT parameter for the database, the process receives the number of global buffers specified by the active USER LIMIT parameter, as shown in Example 4–8.

Example 4–8 Using the RDM$BIND_BUFFERS Logical Name to Specify More Global Buffers Than the Value Specified by the Active USER LIMIT Parameter

```
$! A user executes the following commands:
$ DEFINE RDM$BIND_BUFFERS 50
$!
$ SQL
SQL> ATTACH ’FILENAME mf_personnel’;
   .
   .
   .
$!
$! Another user on the same node as the first user executes
$! the following command:
$ RMU/SHOW USERS mf_personnel
Oracle Rdb V7.0-00 on node TRIXIE 28-MAY-1996 10:39:00.12
database _$111$DUA176:[LOGAN.V70]MF_PERSONNEL.RDB;1
    - First opened 28-MAY-1996 10:20:53.50
    * database is opened by an operator
    - current after-image journal file is
_$111$DUA176:[LOGAN.V70]RICK1.AIJ;1
    - global buffer count is 250
    - maximum global buffer count per user is 25
    - 225 global buffers free
    - 1 active database user
    - 20808D50:1 - _RTA7: - non-utility, RICK - active user
        - image $111$DUA600:[SQL_X07001.VAX.][CODE]SQL$701.EXE;1
        - 25 global buffers allocated
$
```

Note

All buffers in the global buffer pool are available to all users at any given time. Buffer allocations and buffer limits indicate how many buffers a process can use at one time. If, for example, 1000 buffers are in the global buffer pool, and 100 users are each allocated 10 buffers, each user can read any of the 1000 buffers without performing any disk I/O, but each user can read or modify only 10 buffers at any one time. The 10 buffers that a user can read or modify at any one time are the user’s allocate set.

Decide the maximum number of global buffers a process can allocate by dividing the total number of global buffers set by the NUMBER IS qualifier by the total number of processes you want to have *guaranteed* database access. For example, if the total number of global buffers is 200 and you want to guarantee at least 10 users access to the database, set the maximum number of global buffers for each process to 20 (USER LIMIT IS 20). Section 4.1.2.9 suggests a method for determining the maximum number of global buffers a process should be allowed to allocate. Note that you can set the value of NUMBER OF BUFFERS equal to 10, thus allowing 20 users to attach to the database for normal processing. But, by setting USER LIMIT equal to 20, you enable special applications that request more buffers than the default (through use of the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter) to allocate up to 20 buffers.

The value specified by the USER LIMIT parameter cannot be greater than the total number of global buffers. The default value is 5 buffers.

The USER LIMIT parameter is important to operations and system performance. The larger the value, the fewer processes are guaranteed access to the database. The value specified by the USER LIMIT parameter determines the maximum number of buffers a process can allocate regardless of the following values:

- Default number of local buffers to be used by a process, set with the NUMBER OF BUFFERS parameter to the SQL CREATE DATABASE statement
- Number of buffers defined by RDM$BIND_BUFFERS or RDB_BIND_BUFFERS

4.1.2.9 Tuning Global Buffers

You can determine the amount of virtual memory reserved for a database global buffer pool on a node by multiplying the values of the BUFFER SIZE and NUMBER IS parameters.

The size of the buffer pool is critical to system performance.

- If the buffer pool is too large, the operating system is forced to perform virtual paging. In this case, two I/Os might be needed to read a database page into memory: one by the operating system to page fault the buffer into the working set and one by Oracle Rdb to read the page into the buffer pool.

OpenVMS VAX OpenVMS Alpha

If you suspect that the buffer pool is too large, use the SHOW SYSTEM command to see how much paging the system is experiencing. ♦

- If the buffer pool is too small, Oracle Rdb has to perform more database I/O. Use the Performance Monitor's PIO Statistics–Data Writes, PIO Statistics–Data Fetches, and PIO Statistics–SPAM Fetches screens to check for excessive I/O. See Section 4.1.1.2, Section 4.1.1.3, and Section 4.1.1.4, respectively, for information on these displays.

The remainder of this section describes how to choose values for the NUMBER IS, USER LIMIT, and NUMBER OF BUFFERS parameters. For maximum performance in a multinode system, you should tune the NUMBER IS and USER LIMIT global buffer parameters on each node in the cluster using the RMU Open Global_Buffers=(Total=n,User_Limit=t) command.

Note

The default values for the global buffer parameters are unlikely to provide the optimum performance for your database. It is not possible to supply default values that suffice for the many possible database and system configurations.

- NUMBER IS

  The following factors affect the value you select for the total number of global buffers:

  - Amount of physical memory available
  - Percentage of physical memory that can be allocated to Oracle Rdb for the database

  For example, if 128 megabytes of physical memory are available, and 10 percent of this memory can be allocated for your database, and buffer size is 3 kilobytes, then:

$$NUMBER\ IS = \frac{128 * .10}{.003} = \frac{12}{.003} = 4000$$

  This system can support a global buffer pool of 4000 buffers.

- USER LIMIT

  The factor for this parameter is: how many users need guaranteed database access? To continue the example, if you want to guarantee access for 50 users, then:

$$USER\ LIMIT = \frac{4000}{50} = 80$$

This system guarantees database access for a minimum of 50 users, and each user can allocate 80 buffers.

- NUMBER OF BUFFERS

  The factor for this parameter in a system using global buffers is: how many users need access during times of peak database activity? To continue the example, if there are 200 users at peak time, then:

$$NUMBER\ OF\ BUFFERS = \frac{4000}{200} = 20$$

  This system allows 200 users to allocate a default of 20 buffers apiece.

4.1.2.10 Benefits of Global Buffer Overflow Management

A major benefit of enabling global buffers in an environment where data is shared among concurrent users is that a user has read access to all global buffers defined database-wide on a particular node. If a user needs to read a row from a new page, then a disk I/O can be avoided if the page already exists in a global buffer in the pool for the database on that node, and the buffer is shared. Because a particular page can be shared in the global buffer pool, disk I/O is reduced. Less memory is used because fewer buffers are necessary.

A database process is allowed to write to and control the number of global buffers in its allocate set. When a user reads a page, the global buffer that contains the page becomes part of the user's allocate set.

The maximum number of global buffers a user can write to and control is specified by the active value for the USER LIMIT parameter. This parameter specifies the maximum number of global buffers that can be allocated to a database user at attach time by the Oracle Rdb monitor. The default number of global buffers allocated to a user is specified by the NUMBER OF BUFFERS parameter. The actual number of global buffers allocated to a process is dynamic and can be set on a per-user basis with the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter. If RDM$BIND_BUFFERS or RDB_BIND_BUFFERS is defined, then this value overrides the default as long as the RDM$BIND_BUFFERS or RDB_BIND_BUFFERS value is not higher than the active USER LIMIT value.

It is important to note that for a database attach to be successful, the appropriate number of buffers to be allocated must be available from the database global buffer pool at attach time. If there are insufficient global buffers to allocate, then the attach will fail.

As stated previously, a database process is allowed to write to and control the number of global buffers in its allocate set. However, if some global buffers are not allocated to a user's allocate set, a user may effectively buffer data in global buffers in excess of the allocate set.

The reason for this behavior is due to a modified version of the least-recently used (LRU) buffer replacement algorithm for choosing victim buffers in the case of buffer overflow, as described in the following text. This situation is possible when the total number of buffers allocated to users is less than the total number of global buffers defined for the database.

For example, assume that the number of database-wide global buffers is specified as 2,000 buffers with a USER LIMIT of 100 buffers (allowing a maximum of 20 concurrent attaches with maximum allocate sets). If 10 users are concurrently attached with maximum allocate sets, then only 50% or 1,000 global buffers are allocated to users, leaving 1,000 apparently "unused" global buffers. However, the number of buffers that contain data accessible by these users can easily exceed the 1,000 allocated buffers.

When a user's buffer pool is full and a request is made to bring in another page, Oracle Rdb uses a least-recently used (LRU) algorithm to choose a victim buffer to make room for the new page. Pool overflow is handled differently with global buffers and local buffers.

With local buffers, the data in the victim buffer is lost from the buffer pool and is replaced by the new buffer. If the user needs to read the data that was in the victim buffer again, the data must be read again from disk.

With global buffers, if the user's allocation of global buffers is full and a new page needs to be brought into the user's allocate set, the benefit is that the victim buffer may remain in memory after replacement. To make room in the user's allocate set for the new buffer that contains the desired page, a victim buffer is chosen in the allocate set using the LRU buffer replacement algorithm, but with global buffers, the victim buffer is not necessarily lost from the global buffer pool.

An unallocated, empty buffer in the global buffer pool will always be chosen first to receive a new buffer of data, followed by a least-recently used, unallocated full buffer. Oracle Rdb uses a pseudo-LRU algorithm to choose the receiving buffer in the global buffer pool. If an unallocated buffer is available from the database global buffer pool, then it receives the new buffer. In this case, the user's internal global buffer control structures are updated to point to the location of the new buffer in memory, but the victim buffer previously pointed to by the user structures remains in global memory. If the user needs to reread the data that was in the victim buffer (which is currently not part of the user’s allocate set), then the data may not need to be read from disk because it may still reside in the global buffer pool for the database.

However, if all global buffers are allocated to users, then the victim buffer is overwritten with the new buffer, and the information in the victim buffer is lost, not only from the user’s allocate set, but also from the global buffer pool. If the user needs to read the data that was in the victim buffer again, then in this case, the data would need to be read again from disk.

In a single-user environment, the user effectively writes to all global buffers allocated to the database due to the way in which Oracle Rdb manages the global buffer pool and handles user buffer overflow. In a multiuser environment, because the user owns or controls only the buffers in the allocate set, it is less likely that an unallocated buffer will be found to accommodate the new buffer. However, it is more likely that the desired buffer is already in memory (due to a read by another user).

If the whole database fits in the global buffer pool, but exceeds the size of the user’s allocate set, then the entire database can be memory resident due to this beneficial effect! Once the database is read into global buffers, all subsequent reads take place from memory; in read/write environments, only writes to disk need to take place. The benefits from an entirely memory-resident database should be obvious; this is a significant feature of the Oracle Rdb global buffer implementation. Similarly, this benefit can be experienced in an environment where a subset of the database that is typically accessed fits in the global buffer pool. Such an environment may be a manufacturing facility, where historical data is kept on line, but where users typically access only current data.

Example 4–9 shows how read operations can be saved in a database with global buffers. In Example 4–9, assume that the page size is equal to the buffer size, that both users are attached to the database simultaneously, and that no page fetches have occurred prior to the example. For simplicity, metadata buffers are excluded from the pool.

In Example 4–9, the following global buffer parameters are in effect for the database:

- Global buffers are enabled.
- The global buffer count (NUMBER IS parameter) is 6.
- The max global buffer count per user (USER LIMIT parameter) is 3.
- The default buffers per user (NUMBER OF BUFFERS parameter) is 2.
- The two users, User 1 and User 2, are on the same node, N1.

- User 2 defines RDM$BIND_BUFFERS or RDB_BIND_BUFFERS to be 3.
- User 1 uses 2 buffers and User 2 uses 3 buffers.

The example shows seven steps that take place, along with comments that describe what occurs with each step.

Example 4–9 Saving Read Operations in a Database with Global Buffers Enabled

```
Step  User 1   User 2        Global Buffer Page Walk-Through
      Page     Page                    Comments
      Fetch    Fetch
-------------------------------------------------------------------
1       1               User 1 reads page 1.

2       2               User 1 reads page 2.

3       3               Page 1 is the victim, User 1 reads page 3
                        into a new, unallocated buffer.  Page 1
                        stays in the global buffer pool.

4       1               Page 2 is the victim, User 1 finds page 1
                        in the global buffer pool, saving a read
                        operation.  Page 2 stays in the global
                        buffer pool.

5                1      User 2 finds page 1 in the global buffer
                        pool, saving a read operation.

6       2               Page 3 is the victim and remains in the
                        global buffer pool.  User 1 finds page 2
                        in the global buffer pool and saves a
                        read operation.

7                3      User 2 finds page 3 in the global buffer
                        pool and saves a read operation.
```

Example 4–9 shows how global buffers reduce I/O to disk in a multiuser environment where users share data, and how global buffer management of victim buffers results in additional disk I/O savings.

This example shows that Oracle Rdb buffers data in global buffers in excess of the user’s allocate set due to the way in which victim buffers are handled. By increasing the size of the database’s global buffer pool, a reduction in direct I/O occurs, because more of the data requested is memory resident in the global buffer pool. Until the entire database is memory resident in the global buffer pool, some data is lost from the global buffer pool, resulting in additional direct I/O to bring back needed pages.

The point of this discussion is not to recommend the use of smaller allocate sets. The size of the user allocate set is critical to the performance of your application and the effectiveness of your buffer cache, especially in multiuser environments. In read-intensive environments, cache effectiveness reduces the need to go to disk for the desired data. In update-intensive environments, the frequency of disk writes is also influenced by the size of the user's buffer pool, because marked (or updated) buffers are flushed to disk in a batch-write operation in the event of buffer overflow. Using an insufficient allocate set would result in more frequent batch-write operations to disk, diminishing the beneficial effects of the batch-write optimization.

4.1.2.11 Benefits of Data Persistence in Global Buffer Memory

If an Oracle Rdb database is explicitly opened (and left open) with the RMU Open command, then data that exists in Oracle Rdb global buffers as a result of database access will persist in memory across database attaches. This results in significant I/O savings if data is accessed again between attaches to the database. For example, after an application program is run initially, in which the data is read and buffers loaded, many (if not all) of the desired database pages reside in memory. This means fewer data file reads are reported by the Performance Monitor for each iteration after the initial run, because more of the data needed by the application already exists in global buffers.

To take advantage of this benefit, you must explicitly open the database (and keep it open) with the RMU Open command. The persistence of data in global buffer memory across database attaches can also mean faster application startup and response time.

4.1.2.12 Modifying Parameters When Global Buffers Are Enabled

When local buffers are enabled for a database, the Oracle Rdb monitor creates a global section or shared memory partition to maintain the database root data structures with the first attach to the database. When global buffers are enabled for a database, the global section or shared memory partition is extended to include the global buffer pool and the data structures required to maintain the integrity of its resident data. These data structures are used to maintain the following information:

- Global buffer control blocks
- Allocate set block counted lists
- Process local lock for system-owned database and storage area locks

Because of these extra data structures used with global buffers, you may need to modify some existing system parameters if you decide to enable global buffers for a database. You can use the Performance Monitor Buffer Information screen to display the size of the global section or shared memory partition when global buffers are disabled and when global buffers are enabled for a database. When you know the size of a database's global section or shared memory partition, you can use that information to help you determine whether or not you need to modify existing system parameters.

The Buffer Information screen displays values that provide an *estimated* size of the global section or shared memory partition. Example 4–10 shows a Buffer Information screen.

Example 4–10 Using the Buffer Information Screen to Determine the Size of a Global Section for a Database

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 21-JUN-1996 10:36:06
Rate: 3.00 Seconds             Buffer Information          Elapsed: 01:32:15.48
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------

Default user buffer count is 20
Default recovery buffer count is 20
Buffer size is 6 blocks
Global Buffers are enabled
- Global buffer count is 1000
- Maximum global buffer count per user is 40
- Page transfer via memory is disabled
Global section size with global buffers disabled is 548934 bytes  <----- Notice
- With global buffers enabled is 4078056 bytes                    <-----
Asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 10 buffers
Detected asynchronous pre-fetch is enabled
- Maximum pre-fetch depth is 4 buffers
- Pre-fetch threshold is 4 pages
Asynchronous batch-write is enabled
- Clean buffer count is 4
- Maximum batch size is 4 buffers
--------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
```

The first value (with global buffers disabled) indicates the size of the database's global section when global buffers are disabled for the database. The second value (with global buffers enabled) indicates the size of the global section when global buffers are enabled for the database. These two values are not stored in the database root file, but instead are calculated from information that is stored in the root file. You can use the Buffer Information screen to determine the size of the global section that will be required when global buffers are enabled for your database *before* you actually enable global buffers.

With global buffers disabled for a database, set the BUFFER SIZE, NUMBER IS, USER LIMIT, and NUMBER OF BUFFERS parameters to what you will use when global buffers are enabled for the database. Also, set the other database parameters to what you plan to use for the database when global buffers are enabled. Then use the Buffer Information screen. The global section size values will help you to determine whether the increased size of the global section when global buffers are enabled means that you need to increase system parameters and process quotas *before* you actually enable global buffers.

In Example 4–10, the Buffer Information screen is used to determine the number of bytes required for the global section of a database with 1000 global buffers, a limit of 40 global buffers per user, and a buffer size of 6 blocks.

The global section size values show that for this database the global section will be more than seven times larger when global buffers are enabled than when global buffers are disabled.

The size of the global section is provided in bytes instead of pages, because the page size can differ from operating system to operating system (for example, 512 bytes per page for OpenVMS, and possibly a different number of bytes per page for other operating systems). To determine the number of pages for the global section of your database, use this calculation:

```
derived number of bytes / bytes per page = number of pages for global section
```

The following example shows how to use the global section size values to compute the number of pages for the global section when global buffers are enabled for the mf_personnel database in Example 4–10:

```
4078056 / 512 = 7964.9
```

The calculated value for the number of pages for the global section or shared memory partition should be rounded up to the next integer value; for the database in this example, the estimated size of the global section when global buffers are enabled is 7965 pages. You should also add 10 to 15 pages to the estimated value to ensure that there are enough pages for the global section. For this example, assume that a global section of 7980 pages will be needed when global buffers are enabled for the database.

Note that you must take the database off line if you decide to enable global buffers.

OpenVMS VAX OpenVMS Alpha

You must shut down and reboot the system to change the OpenVMS SYSGEN parameters. Oracle Corporation recommends that any modifications to the SYSGEN parameters be made through the MODPARAMS.DAT file and the OpenVMS AUTOGEN facility. It is possible to hang a system on reboot due to hard-coded parameters.

Also, note that these modifications may require you to change the Oracle Rdb monitor process account quotas to ensure the paging file quota is adequate. See the explanation of the GBLPAGFIL parameter later in this section for more information.

The rest of this section explains how to determine appropriate settings for the GBLSECTIONS, GBLPAGES, GBLPAGFIL, VIRTUALPAGECNT, and PGFLQUOTA parameters when global buffers are enabled for a database.

The GBLSECTIONS Parameter

The GBLSECTIONS parameter determines the total number of global sections that can be created on the system. One global section is allocated for the database root on your system when the database is first opened, whether global buffers are enabled for the database or not. Because a database uses one global section whether global buffers are enabled or not, you do not need to modify the GBLSECTIONS parameter when you enable global buffers.

The GBLPAGES Parameter

The GBLPAGES parameter determines the total number of global pages that can be created on the system. This and VIRTUALPAGECNT are probably the most critical parameters. See the explanation of the VIRTUALPAGECNT parameter later in this section for more information.

The GBLSECTIONS, GBLPAGES, GBLPAGFIL, and VIRTUALPAGECNT parameters are modifiable; however, because of their nondynamic nature, any change requires a system reboot.

You can verify the use of the GBLPAGES and GBLSECTIONS parameters using the OpenVMS System Generation utility (SYSGEN), the OpenVMS Install utility (INSTALL), and lexical functions. Issuing the SYSGEN SHOW GBLPAGES and SHOW GBLSECTIONS commands provides the current number of GBLPAGES and GBLSECTIONS entries on the system.

```
$ MCR SYSGEN
SYSGEN>  SHOW GBLPAGES
Parameter Name            Current    Default     Min.     Max.     Unit  Dynamic
--------------            -------    -------    -------  -------   ----  -------
GBLPAGES                   350000      10000       512       -1 Pages
SYSGEN>  SHOW GBLSECTIONS
Parameter Name            Current    Default     Min.     Max.     Unit  Dynamic
--------------            -------    -------    -------  -------   ----  -------
GBLSECTIONS                  2500        250        60     4095 Sections
```

The current number of GBLSECTIONS entries is 2500 and the current number of GBLPAGES entries is 350000.

Issuing the LIST/GLOBAL/SUMMARY command at the INSTALL prompt determines the number of global sections and global pages used.

```
$ INSTALL
INSTALL> LIST/GLOBAL/SUMMARY

        Summary of Local Memory Global Sections
    1022 Global Sections Used,  132790/217210 Global Pages Used/Unused
```

This display shows that 1022 global sections are used out of the total of 2500 that SYSGEN has on the system, leaving 1478 global sections available. Out of the total of 350000 global pages that SYSGEN has on the system, 132790 global pages are used, leaving 217210 global pages available.

You can also use the lexical function F$GETSYI to get the number of available GBLPAGES and GBLSECTIONS entries. Define the following symbols and invoke them at the DCL level or in interactive SQL to display the number of available GBLPAGES and GBLSECTIONS entries.

```
$ GBLPAGES    :==       "write sys$output f$getsyi(""free_gblpages"")
$ GBLSECTIONS :==       "write sys$output f$getsyi(""free_gblsects"")
```

With these symbols defined, you can check how many GBLPAGES and GBLSECTIONS are available on your system:

```
$ GBLPAGES
217210
$ GBLSECTIONS
1478
```

Note that the values returned using the lexical function F$GETSYI are the same values for available global pages and global sections that were displayed with the INSTALL/LIST/GLOBAL/SUMMARY command.

Before altering your database to enable global buffering, perform a simple database attach with local buffers enabled. This shows that you have allocated a global section and a number of global page entries to the database root structures:

```
$ SQL
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SHOW DATABASE *
   .
   .
   .
        Number of users:              50
        Number of nodes:              16
        Buffer Size (blocks/buffer):   6
        Number of Buffers:             20
        Number of Recovery Buffers:    20
        Snapshots are Enabled Immediate
        Carry over locks are enabled
        Lock timeout interval is 0 seconds
        Adjustable lock granularity is enabled
        Global buffers are disabled (number is 250, user limit is 5)
   .
   .
   .
SQL> $ GBLPAGES
217122
SQL> $ GBLSECTIONS
1477
```

The "$ GBLPAGES" shows that the number of available global page entries dropped from 217210 to 217122 (88 entries have been allocated). The "$ GBLSECTIONS" shows that the number of available global sections entries dropped from 1478 to 1477 (one global section has been allocated). One global section is allocated when a database is opened.

In the next example, global buffers are enabled for the database and 1000 global buffers with a buffer size of 6 blocks are allocated:

```
$ SQL
SQL> ALTER DATABASE FILENAME mf_personnel
cont> GLOBAL BUFFERS ARE ENABLED
cont> (NUMBER IS 1000, USER LIMIT 40);
SQL> --
SQL> ATTACH 'FILENAME mf_personnel';
SQL> --
SQL> SHOW DATABASE *
   .
   .
   .
        Number of users:               50
        Number of nodes:               16
        Buffer Size (blocks/buffer):   6
        Number of Buffers:             20
        Number of Recovery Buffers:    20
        Snapshots are Enabled Immediate
        Carry over locks are enabled
 Lock timeout interval is 0 seconds
 Adjustable lock granularity is enabled
 Global buffers are enabled (number is 1000, user limit is 40)
   .
   .
   .
```

```
SQL> --
SQL> $ GBLPAGES
210262
SQL> $ GBLSECTIONS
1477
```

Before attaching to the database, the number of global page entries stood at 217210 (before global buffers were enabled for the database). After attaching to the database, the number of GBLPAGES dropped from 217210 to 210262 available entries; 6948 global pages are allocated now for the global section.

Finally, increase the maximum number of users to 75 and enable after-image journaling for the database:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>    NUMBER OF USERS IS 75
cont>    JOURNAL FILENAME SQL_DISK1:[RICK]MF_PERS_JOURNAL
cont>    JOURNAL ALLOCATION IS 500 BLOCKS
cont>    JOURNAL EXTENT IS 100 BLOCKS;
%RDMS-W-DOFULLBCK, full database backup should be done to ensure future recovery
SQL> --
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SHOW DATABASE *
   .
   .
   .
        Number of users:               75
        Number of nodes:               16
        Buffer Size (blocks/buffer):   6
        Number of Buffers:             20
        Number of Recovery Buffers:    20
        Snapshots are Enabled Immediate
        Carry over locks are enabled
 Lock timeout interval is 0 seconds
 Adjustable lock granularity is enabled
 Global buffers are enabled (number is 1000, user limit is 40)
   .
   .
   .
        AIJ File Allocation:           500
        AIJ File Extent:               100
   .
   .
   .
SQL> --
SQL> $ GBLPAGES
209242
SQL> $ GBLSECTIONS
1477
```

The number of GBLPAGES dropped from 217210 to 209242 available entries; 7968 global pages are allocated now for the global section. Note that the database in this example has the same global buffer and database characteristics as the database in Example 4–10. The global section size values from the Buffer Information screen estimated a global section size of 7965 global pages for the database in Example 4–10 with global buffers enabled. This example shows that when global buffers are enabled for this database, the actual size of the global section is 7968 global pages.

It is difficult to provide a simple formula that will specify how many GBLPAGES entries you need. As you can see from the examples in this section, the size of the global page pool allocated when global buffers are active depends on many factors, such as the number of global buffers, the maximum number of global buffers per user, the number of users, the number of storage areas in the database, whether after-image journaling is enabled, and so forth. These factors and others make it difficult to provide a simple formula that you can use. Using the global section size values from the Buffer Information screen is the best way to estimate the number of global pages that will be required for the global section (remember to add 10 to 15 global pages to the estimate to ensure enough pages for the global section).

The number of users is one database parameter the database administrator (DBA) has control over. Often, the DBA leaves certain database parameters set as default values because these values may be acceptable and have had no real impact on system resources or performance. When a database is created with no explicit total number of users parameter specified, the default is set to 160. Most applications do not require that high a value, especially applications running on a single node, or applications running with a transaction processing monitor.

For systems where memory is a critical resource, the DBA may want to manually adjust the total number of users to a more realistic value and conserve memory.

The following table shows how the number of global pages allocated increases as the number of users increases:

```
Request made for 500 buffers at 6 blocks each

Number of users      Global pages allocated.

      20                   3488
      60                   3580
      80                   3626
     100                   3674
     200                   3904
     300                   4136
     400                   4366
     500                   4598
```

After setting the appropriate global buffer and database parameters with global buffers disabled, use the global section size values from the Performance Monitor Buffer Information screen to estimate the number of global pages that will be required for your database's global section. Also, it is important to remember that the Buffer Information screen estimates the size of a global section based on information stored in the database root file. If you use the RMU Open command with the Global qualifier to specify different global buffer parameters than those stored in the database's root file, the size of the database's global section is likely to decrease or increase.

Note that on each node from which a database is opened, Oracle Rdb maps a global section for that database. Therefore, you need to check the requirements for the GBLSECTIONS, GBLPAGES, GBLPAGFIL, VIRTUALPAGECNT, and PGFLQUOTA parameters on each node from which a database will be opened.

The GBLPAGFIL Parameter

The SYSGEN GBLPAGFIL parameter defines the maximum number of global pages with page file backing store that can be created on a system. Determining a value for GBLPAGFIL depends on many factors, including the number of databases, the number of run units, the number and size of each global buffer, and the overhead.

The best way to determine the number of GBLPAGFIL entries that you need for a database is to use the Performance Monitor Buffer Information screen to find the number of global pages required for the database's global section. The number of GBLPAGFIL entries that you need for the database will be the same as the number of global pages required for the database.

If you use more than one database at a time, calculate the requirement for each database. If you change the GBLPAGFIL parameter, you must reboot your system for the change to take effect.

If the value for the GBLPAGFIL parameter is too low, Oracle Rdb does not allow you to open a database with global buffers. In this situation, you will encounter the following errors:

```
%RDMS-F-CANTOPENDB, database could not be opened as requested
-RDMS-F-CANTCREGBL, error creating and mapping database global section
-SYSTEM-F-EXGBLPAGFIL, exceeded global page file limit
%RMU-W-FATALERR, fatal error on <db-name>
```

Because the global section created by Oracle Rdb will form a global page file section, the number of global pages used by the section will be decremented from the GBLPAGFIL system value. Therefore, enough GBLPAGFIL must remain to accommodate the number of global pages required by the database global section.

You can use the System Dump Analyzer (SDA) utility to determine how many GBLPAGFIL entries are available. You must have the OpenVMS CMKRNL privilege to use SDA. The CMKRNL privilege is powerful; it should not be given to users who do not need it. The following example shows how to use SDA to determine how many GBLPAGFIL entries are available:

```
$ ANALYZE/SYSTEM
SDA> !
SDA> ! Examine the original SYSGEN setting
SDA> !
SDA> EVALUATE @SGN$GL_GBLPAGFIL
Hex = 00002FA8   Decimal = 12200     UCB$M_TEMPLATE+00FA8
SDA> !
SDA> ! Examine the available GBLPAGFIL
SDA> !
SDA> EVALUATE @MMG$GL_GBLPAGFIL
Hex = 000015A1   Decimal = 5537     UCB$M_UNLOAD+005A1
```

In this case, any database that required more than 5537 global pages could not be opened, and would result in the errors detailed previously.

You can use the RMU Open command to reduce the number of global buffers to a reasonable amount:

```
$ RMU/OPEN/GLOBAL=(TOTAL=10,USER_LIMIT=5) <db-name>
```

You can then use the SQL ALTER DATABASE statement to specify a smaller number of global buffers for the global buffer pool. The value specified is stored in the database root file:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>    GLOBAL BUFFERS ARE ENABLED
cont>    (NUMBER IS 10);
```

The ALTER DATABASE statement also allows you to disable global buffers:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>    GLOBAL BUFFERS ARE DISABLED;
```

If you need to increase the number of GBLPAGFIL entries, you may also need to increase the size of your page file or add a page file. You can make these changes using the OpenVMS System Generation utility (SYSGEN). You must then follow up with the necessary changes to the system parameters using the MODPARAMS.DAT file and the AUTOGEN command procedure.

The VIRTUALPAGECNT Parameter

The VIRTUALPAGECNT parameter sets the total number of virtual pages that your process is allowed to map. Because your process maps to the global section, as the global section grows, so should the number of virtual pages that processes are allowed to map. Therefore, if you increase the GBLPAGES parameter, you should increase the VIRTUALPAGECNT parameter by about the same amount.

If you change the VIRTUALPAGECNT parameter, you must reboot your system for the change to take effect.

Oracle Rdb users who have global buffers enabled on the database sometimes encounter the -LIB-F-INSVIRMEM, insufficient virtual memory error. This error message indicates that the VIRTUALPAGECNT or PGFLQUOTA quotas for a process are not large enough.

When global buffers are enabled, the monitor process' virtual memory consumption is proportional to the number of global buffers.

In addition, the more global buffers that are defined for the database, the higher the virtual memory requirements are for each user. Each user needs virtual memory to map the buffer pool.

Note

Although this section discusses the virtual memory consumed by the Oracle Rdb monitor, keep in mind that virtual memory and page file quota limits also apply to user processes. This section discusses the monitor because it is the less obvious place to look for a quota problem.

The monitor does not completely release all virtual memory when a database with global buffers enabled is closed, and this appears to contribute to the frequency of virtual memory errors.

Use the following steps to diagnose insufficient virtual memory problems:

1. Get the process identifier (PID) of the Oracle Rdb monitor.

   If you are experiencing insufficient virtual memory errors, first get the PID of the RDMS_MONITOR process:

```
$ SHOW SYSTEM
VAX/VMS V5.5-2  on node GNPIKE  28-MAY-1996 11:17:18.88   Uptime  8 04:32:17
  Pid    Process Name    State  Pri      I/O       CPU       Page flts Ph.Mem
20800201 SWAPPER         HIB     16        0   0 00:09:20.18         0      0
20800206 CONFIGURE       HIB     10      204   0 00:00:03.39       123    320
20800219 RDMS_MONITOR70  LEF     15      143   0 00:00:05.40     18238    195
```

2. Get accounting information for the monitor.

   Use the SHOW PROCESS/ACCOUNTING command to see how long the process has been connected, the user information, and the peak virtual size.

```
$ SHOW PROCESS /ACCOUNTING /ID=20800219

28-MAY-1996 11:21:14.87   User: SYSTEM          Process ID:   20800219
                          Node: GNPIKE          Process name: "RDMS_MONITOR70"

Accounting information:
 Buffered I/O count:        44  Peak working set size:       1501
 Direct I/O count:          99  Peak virtual size:         139072
 Page faults:            18261  Mounted volumes:                0
 Images activated:           0
 Elapsed CPU time:          0 00:00:05.41
 Connect time:              8 04:18:51.37
```

3. Use the information to diagnose the problem.

   The problem could be one of the following:

   - Virtual page size is too small

     If the peak virtual size is at or near the limit for VIRTUALPAGECNT in SYSGEN, you will probably need to increase VIRTUALPAGECNT using AUTOGEN and reboot the system.

```
$ MCR SYSGEN
SYSGEN>  SHO VIRTUAL
Parameter Name            Current    Default     Min.     Max.     Unit  Dynamic
--------------            -------    -------    -------  -------   ----  -------
VIRTUALPAGECNT             139072       9216       512   1200000 Pages
```

     VIRTUALPAGECNT determines the maximum number of pages that can be mapped for a process. The peak virtual size returned by the SHOW PROCESS/ACCOUNTING command displayed how many pages are currently mapped for the process.

In this example, the Current VIRTUALPAGECNT setting in SYSGEN is 139072. The peak virtual size is 139072. The monitor is hitting the virtual memory limit set through VIRTUALPAGECNT.

The Current column in the SYSGEN display determines the values in use by the system. Default, Min., and Max. provide information on the valid range of values for the parameter. SYSGEN values are not dynamic like working sets. The value for the Current column is fixed until it is manually changed. Dynamic SYSGEN parameters take effect right away. Nondynamic parameters take effect the next time the system is rebooted.

- The monitor’s page file quota is too small

  The most common cause of the insufficient virtual memory problem occurs when the PGFLQUOTA value is too small on the account from which the monitor was started. By default, the monitor is started from the SYSTEM account. The SYSTEM account normally has PGFLQUOTA values that are much too small for more than a few thousand global buffers. PGFLQUOTA determines the maximum number of pages that the process can use in the system paging file to manage the process’ virtual memory. When PGFLQUOTA is exhausted, the system has no place to put any additional mapped pages, so an error occurs. The monitor log file reports errors where database attaches were disallowed because of quota problems with the monitor process.

  If your SYSGEN VIRTUALPAGECNT is much bigger than the peak virtual size shown for the monitor with the SHOW PROCESS /ACCOUNTING command, you are probably hitting this limit. The following example shows how to increase the PGFLQUOTA parameter for the monitor account:

```
$ SET DEF SYS$SYSTEM
$ MCR AUTHORIZE
UAF> MODIFY SYSTEM/PGFLQUO=400000
%UAF-I-MDFYMSG, user record(s) updated
UAF>
```

  Another method to ensure that the monitor process has the appropriate quotas is by using the RDM$MON_USERNAME logical name. The logical name RDM$MON_USERNAME designates the name of the user whose quotas the monitor process, upon startup, is to inherit. See Section A.78 for more information on the RDM$MON_USERNAME logical name.

Make sure your page files are big enough to support the new size. Then restart the monitor (either from the system account or after defining the RDM$MON_USERNAME logical name to be an account with the appropriate quotas for the monitor). This causes the new page file quota to be used.

You will also run into problems if your user process exceeds the virtual memory quota or has an insufficient page file quota.

Sometimes, correcting the virtual memory problem allows other resource limitation problems to be reported:

```
$ RMU/OPEN SQL$DATABASE/GLOBAL=(TOTAL=25000,USER=10)
%RDMS-F-CANTOPENDB, database could not be opened as requested
-RDMS-F-CANTCREGBL, error creating and mapping database global section
-SYSTEM-F-EXGBLPAGFIL, exceeded global page file limit
```

This error message indicates that you need to increase the GBLPAGFIL parameter in SYSGEN. See the explanation of the GBLPAGFIL parameter earlier in this section for more information on adjusting the GBLPAGFIL value.

The PGFLQUOTA Parameter

The PGFLQUOTA parameter determines the maximum number of pages that the process can use in the system paging file to manage the process' virtual memory. When PGFLQUOTA is exhausted, the system has no place to put any additional mapped pages, so an insufficient virtual memory error occurs. The most common cause of the insufficient virtual memory problem occurs when the PGFLQUOTA value is too small on the account from which the monitor was started. By default, the monitor is started from the SYSTEM account, which normally has PGFLQUOTA values that are much too small for more than a few thousand global buffers.

See the explanation of the VIRTUALPAGECNT parameter earlier in this section for more information on the PGFLQUOTA parameter and analyzing insufficient virtual memory errors. ♦

4.1.2.13 Analyzing Global Buffer Performance

In general, a database with local buffers enabled uses approximately 10% more CPU resources after global buffers are enabled. If CPU is already a limited resource, this additional CPU usage should be considered before you enable global buffers for the database.

When global buffers are enabled, more page faults occur than with local buffers. A higher number of GLOBAL VALID page faults indicates that the page that caused the fault is in the global buffer pool. These are soft faults and are usually harmless.

OpenVMS VAX OpenVMS Alpha If you decide, however, that the number of GLOBAL VALID page faults should be decreased, you can increase the working set quotas for the processes using the database, as described in Section 8.2. The OpenVMS MONITOR PAGE command gives a breakdown of page faults by category.

More lock operations occur with global buffers than with local buffers. Some of the additional lock operations are system-owned locks, which can be costly, while others are local locks. The system-owned page locks maintain page version numbers among nodes in a VMScluster. Each page in a global buffer has a lock associated with it. These system-owned locks do not consume any ENQLM, but for each page that a user has in his or her allocate set, there is a lock charged against ENQLM. The system-owned locks use LOCKIDTBL entries, so LOCKIDTBL (as well as LOCKIDTBL_MAX, SRPCOUNT, and SPRCOUNTV) can be impacted if you have many pages in global buffers. Local inexpensive locks synchronize access to the global buffer data structures. The increase in locks is mostly page locks and some record locks. The database administrator should ensure that LOCKIDTBL, LOCKIDTBL_MAX, SRPCOUNT, and SPRCOUNTV have enough entries to accommodate the extra locking associated with global buffers. Use the OpenVMS MONITOR LOCK command to show the total locks on the system, then verify this total against the LOCKIDTBL entries on the system. ♦

If you want to test the performance of global buffers and local buffers, you should keep the equivalence case in mind. That is, a special case of the settings is when the local and global buffers settings are equivalent. The basis of this equivalence is memory usage, and this can be expanded into the following two rules:

- The total number of buffers used by all the users must be equal for local and global buffers.
- The number of buffers used by each user must be equal for local and global buffers.

One way to set up buffering parameters is as follows:

1. Set up the NUMBER OF BUFFERS parameter (the default number of buffers) to be B.
2. Set up the USER LIMIT parameter (the maximum number of global buffers per user) to be B, also.
3. If you are going to have N users for the performance test, set up the NUMBER IS parameter (the total number of global buffers) to be B times N.

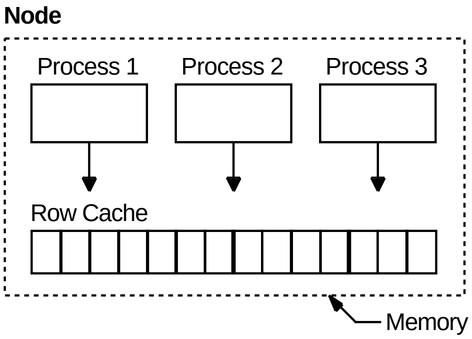
Do not use any buffering logical names or configuration parameters to keep it simple. Following these guidelines permits you to enable and disable global buffers and run performance tests.

4.1.3 Row Caching

A **row cache** is a section of globally accessible memory that contains copies of rows. Row caching provides the ability to store frequently accessed rows in memory, reducing disk I/O. The rows remain in memory even when the associated page has been flushed back to disk. Row caching has the following advantages:

- Reduced database page reads and writes
- Improved response time
- Much lower overhead to access a row in a row cache than for a page in a global or local buffer
- Shorter code path when a row is found in the row cache
- Efficient use of system resources (memory) for shared data

A row cache can contain index structures as well as table data. A row cache is shared by all processes attached to the database as the following shows:



NU−3621A−RA

When you request a row from the database, Oracle Rdb first checks to see if the requested row is mapped to an existing row cache. If a row cache is mapped, Oracle Rdb checks to see if the requested row is in the row cache. If the row is in the row cache, the row is retrieved. If the row is not in the cache, Oracle Rdb checks the page buffer pool. If the row is not in the page buffer pool, Oracle Rdb performs a disk I/O to retrieve the row. The requested row is then inserted into the row cache if it can fit. The code path is shorter when the row is retrieved from a row cache as compared to a page buffer pool or disk.

You can improve performance, for example, by caching the following:

- Heavily accessed data that is shared by multiple users
- Nonleaf nodes of a B-tree index

  Caching the nonleaf nodes effectively pins the index in memory.
- The RDB$SYSTEM storage area

  Caching part or all of the RDB$SYSTEM storage area can improve the performance of metadata queries.

The task of deciding what to cache is easier when the storage areas in your database are already partitioned effectively. The more you know about your data, the more effectively you can determine what table rows and indexes to cache.

There are two types of row caches:

- Physical area

  Physical area caches are defined for a storage area. A physical area cache can contain data from one or more storage areas. You explicitly assign a physical area cache to a storage area with the CACHE USING clause of the CREATE STORAGE AREA or ADD STORAGE AREA clause of the CREATE DATABASE or ALTER DATABASE statement.

  A physical area cache provides a way to cache system records. In addition, when a physical area cache is defined, all rows of different sizes in the specified storage area are candidates for the row cache.
- Logical area

  Logical area caches are defined for a specific table or index. Oracle Rdb automatically assigns a logical area cache to a table or index based on the name.

  The name of a logical area cache must match the name of a table or index that you want to cache.

  You can save a considerable amount of space with a logical area cache by caching rows that are similar in size in one cache.

4.1.3.1 Requirements for Using Row Caches

The following conditions must be true in order to use row caches:

- The number of cluster nodes is one
- After-image journaling is enabled
- Fast commit is enabled
- One or more cache slots are reserved
- Row caching is enabled

Use the RMU Dump Header command to check if you have met the requirements for using row caches. The following command output displays a warning for every requirement that is not met:

```
   .
   .
   .
    Row Caches...
      - Active row cache count is 0
      - Reserved row cache count is 1
      - Sweep interval is 1 second
      - Default cache file directory is ""
      - WARNING: Maximum node count is 16 instead of 1
      - WARNING: After-image journaling is disabled
      - WARNING: Fast commit is disabled
   .
   .
   .
```

4.1.3.2 Enabling Row Caching

You can enable row caching for a database by using the ROW CACHE IS ENABLED clause of the SQL ALTER DATABASE and CREATE DATABASE statements:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ROW CACHE IS ENABLED;
```

You can disable row caching for a database by using the ROW CACHE IS DISABLED clause of the SQL ALTER DATABASE and CREATE DATABASE statements:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ROW CACHE IS DISABLED;
```

Row caching is also disabled if one of the conditions described in Section 4.1.3.1 becomes false.

When row caching is disabled, all previously created and assigned row caches remain in existence for future use when row caching is enabled again.

There must not be any users attached to the database when you enable or disable row caching.

You can verify that row caching is enabled by invoking the Performance Monitor and viewing the main menu. The following example shows a main menu where row caching is enabled:

```
                                Select Display

    A. Summary IO Statistics             O. IO Statistics (by file)  [->
    B. Summary Locking Statistics        P. Locking (one lock type)  [->
    C. Summary Object Statistics         Q. Locking (one stat field) [->
    D. Summary Cache Statistics          R. Lock Statistics (by file)[->
    E. Summary Cache Unmark Statistics   S. Database Parameter Info  [->
    F. Record Statistics                 T. Row Cache (One Cache)    [->
    G. Transaction Duration (Total)      U. Row Cache (One Field)    [->
    H. Custom Statistics                 V. Row Cache Information    [->
    I. Snapshot Statistics               W. Index Information        [->
    J. Process Information      [->      X. General Information      [->
    K. Journaling Information   [->      Y. Objects (one stat type)  [->
    L. Hot Standby Information  [->      Z. Objects (one stat field) [->
    M. IO Statistics            [->      0. Database Dashboard       [->
    N. Global Buffer Information[->      1. Online Analysis & Info.  [->
```

The row caching screens do not appear in the main menu if one or more of the requirements described in Section 4.1.3.1 have not been met.

4.1.4 Gathering Row Cache Information

Use the RMU Dump Header command to display information about row caches, as shown in Example 4–11.

Example 4–11 Row Cache Parameters

```
$ RMU/DUMP/HEADER mf_personnel
   .
   .
   .
  Row Caches...      ❶
     - Active row cache count is 23
     - Reserved row cache count is 54
     - Sweep interval is 1 second
     - Default cache file directory is ""
   .
   .
   .
```

(continued on next page)

Example 4–11 (Cont.) Row Cache Parameters

```
Storage area "EMPIDS_LOW"
   .
   .
   .
    Row Caching...  ❷
      - Row caching is enabled
      - Row cache ID is 1
   .
   .
   .
Row cache "EMPIDS_LOW"
    Cache ID number is 1  ❸
    Allocation...  ❹
      - Row slot count is 1000
      - Maximum row size allowed in cache is 200 bytes
      - Working set count is 10
      - Maximum slot reservation count is 20
      - Row replacement is enabled
    Files... ❺
      - No file directory has been specified
      - File allocation is 100 blocks
      - File extension is 100 blocks
    Shared Memory...  ❻
      - System space memory is disabled
      - Large memory is disabled
      - Large memory window count is 100
    Hashing...  ❼
      - Hash value for logical area DBIDs is 211
      - Hash value for page numbers is 11
    Checkpointing...  ❽
      - Last checkpoint is 0:2
      - Checkpoint sequence is 2

   .
   .
   .
```

The following callouts identify the parameters in Example 4–11:

❶ Summary information

- Active row cache count is 23

  This specifies the number of row caches currently defined in this database.

- Reserved row cache count is 54

  This specifies the number of slots that are available in the database. The cache slots are reserved with the RESERVE n CACHE SLOTS parameter of the ALTER or CREATE DATABASE statements.

- Sweep interval is 1 second

  The sweep interval is specified with the SWEEP INTERVAL parameter of the ROW CACHE clause. A sweep is one full pass through all active row caches, attempting to write modified rows back to their respective storage areas.

- Default cache file directory is ""

  The default cache file directory is the directory where Oracle Rdb places the cache backing store files if you do not explicitly include a directory specification. You can specify a directory location with the following SQL syntax:

  - The LOCATION parameter of the CREATE or ADD CACHE clause
  - The LOCATION parameter with the ROW CACHE clause of the CREATE or ALTER DATABASE statement

  If you do not specify a location, Oracle Rdb places the cache backing store files in the directory of the database root file.

  The backing store files contain information about the contents of the row cache global sections on OpenVMS and shared memory partitions on Digital UNIX.

❷ Storage area information

- Row caching is enabled

  This is specified with the ROW CACHE parameter. Enabling a row cache has no effect unless a row cache is defined and assigned to one or more storage areas. Row caching is disabled by default.

- Row cache ID is 1

  Oracle Rdb assigns an ID to each defined row cache in the database.

❸ Cache ID number is 1

Oracle Rdb assigns an ID to each defined row cache in the database.

❹ Allocation . . .

- Row slot count is 1000

  This is specified with the CACHE SIZE is n ROWS parameter.

- Maximum row size allowed in cache is 200 bytes

  This is specified with the ROW LENGTH is n BYTES parameter.

- Working set count is 10

This is the number of “in use” rows that are not eligible for row replacement.

- Maximum slot reservation count is 20

  This is specified with the NUMBER OF RESERVED ROWS parameter. The default value is 20 rows.

  The number of reserved rows indicates how many slots in the cache Oracle Rdb will reserve for each process. Reserving many rows minimizes row cache locking while rows are inserted into the cache.

- Row replacement is enabled

  This is specified with the ROW REPLACEMENT parameter. Row replacement is enabled by default.

❺ Files . . .

- No file directory has been specified

  The LOCATION parameter specifies a directory specification for the cache backing store file. Oracle Rdb writes to the cache backing store file when the row cache server (RCS) process checkpoints. Oracle Rdb automatically generates a file name with a file extension of .rdc. The default location for the cache backing store file is the directory where the database root file is located.

  The LOCATION parameter can be specified at the database level or at the row cache level. If you include the LOCATION parameter in the ROW CACHE clause, the directory you specify becomes the default directory location for all the row caches that are defined for the database. You can, however, override the default directory location for individual row caches by specifying the LOCATION parameter in the row cache definition.

- File allocation is 100 blocks

  The ALLOCATION parameter specifies the initial size of the .rdc file. The default allocation is 40 percent of the cache size. The cache size is determined by multiplying the number of rows in the cache by the row length.

- File extension is 100 blocks

  The EXTENT parameter specifies the number of pages by which the cache backing store file (.rdc) can be extended after the initial allocation has been reached. The default extent is 127 multiplied by the number of rows in the cache.

❻ Shared Memory . . .

OpenVMS Alpha

- System space memory is disabled

  This is specified with the SHARED MEMORY parameter. This specifies whether Oracle Rdb creates the row cache in shared memory. The row cache is created in a process global section by default. ♦

OpenVMS Alpha

- Large memory is disabled

  This is specified with the LARGE MEMORY parameter. This specifies whether Oracle Rdb creates the row cache in physical memory. Large memory is disabled by default. ♦

- Large memory window count is 100

  This is specified with the WINDOW COUNT parameter. The default value is 100 windows. The WINDOW COUNT specifies how many locations of the physical memory are mapped to each user's private window in virtual address space.

❼ Hashing . . .

- Hash value for logical area DBIDs is 211
- Hash value for page numbers is 11

The hash values are used by Oracle Rdb to fine-tune the distribution of hash table queues in the row cache.

❽ Checkpointing . . .

- Last checkpoint is 0:2

  This specifies the AIJ sequence number and the AIJ virtual block number (VBN) of the last checkpoint.

- Checkpoint sequence is 2

4.1.4.1 How to Create and Use a Row Cache

To use the row caching feature, you must perform the following steps (or accept the Oracle Rdb default values):

1. Reserve slots in the database root file for pointers to row caches.

   When you reserve slots for row caches, you make it possible to add row caches while the database is on line. Note that reserving slots does not enable row caching. See Section 4.1.4.1.1 for more information about reserving row cache slots.

2. Create the row cache.

Use the CREATE CACHE clause of the SQL CREATE DATABASE STATEMENT or the ADD CACHE clause of the SQL ALTER DATABASE or IMPORT statements to create a row cache. You have the option of specifying the number and size of the rows to be stored in the cache. See Section 4.1.4.1.2 for more information about creating a row cache.

3. Choose memory.

   Specify in what area of memory to store the cache. See Section 4.1.4.1.3 for more information.

4. Assign a row cache to a storage area.

   Use the CACHE USING clause of the CREATE STORAGE AREA or ADD STORAGE AREA clauses of the CREATE DATABASE or ALTER DATABASE statements to assign a row cache to a storage area. You only need to assign physical area caches. Oracle Rdb automatically assigns logical area caches. See Section 4.1.4.1.4 and Section 4.1.4.1.5 for more information.

4.1.4.1.1 Reserving Slots for Row Caches When you create a database, you should consider how many row caches your database may need in the future and reserve slots for at least that number of row caches. When you reserve slots, you reserve slots in the database root file for pointers to row caches. If you reserve a sufficient number of slots for row caches, you can add row caches while the database is on line without interrupting database activity.

Use the RESERVE n CACHE SLOTS clause of the ALTER DATABASE statement to reserve slots for row caches, as shown in the following example:

```
SQL> ALTER DATABASE
cont>  FILENAME 'mf_personnel'
cont>  RESERVE 20 CACHE SLOTS;
```

If you do not specify a RESERVE n CACHE SLOTS clause, Oracle Rdb reserves one slot.

4.1.4.1.2 Specifying the Size of a Row Cache When you create a row cache or modify a row cache definition, you have the option of specifying the following:

- Slot size

  The **slot size** is the size of the largest row that can be stored in the row cache. Oracle Rdb will not cache a row if it is too large to fit in the cache. Use the ROW LENGTH IS parameter of the ADD, ALTER, or CREATE CACHE clause to specify the size of the largest row in the cache.

- Slot count

The **slot count** is the number of rows that can be stored in the cache. Use the CACHE SIZE IS parameter of the ADD, ALTER, or CREATE CACHE clause to specify the number of rows that can be stored in the cache.

The following example shows a row cache definition:

```
SQL> ADD CACHE RCACHE_1
cont> ROW LENGTH IS 200 BYTES
cont> CACHE SIZE IS 3000 ROWS;
SQL> --
SQL> -- In this example, the slot size is 200
SQL> -- and the slot count is 3000.
SQL> --
```

If you do not specify the ROW LENGTH clause, Oracle Rdb creates a cache with rows that can hold up to 256 bytes and if you do not specify the CACHE SIZE clause, Oracle Rdb creates a cache that can contain up to 1000 rows.

Oracle Rdb automatically rounds up the row length to the next 4-byte boundary, if the value specified is not divisible by four. This is done because longword aligned data structures are optimal for performance.

It is extremely important to select a proper size for the row cache. As stated previously, if a row is too large, Oracle Rdb will not cache the row. System performance is adversely affected because Oracle Rdb always checks the cache for the row before retrieving the row from disk. Use the RMU Dump Areas command to determine the sizes of the data rows, hash buckets, and B-tree nodes. Keep in mind that row sizes within a table can vary greatly. If, for example, the largest row stored in a table is 100 bytes, but the majority of the rows range between 40 and 50 bytes, you may not necessarily want to choose 100 bytes for the slot size. However, you should account for most of the rows, including overhead. If you automatically select the largest row size without comparing it to the sizes of the other rows in the table, you could waste a lot of memory.

The following example dumps a few pages from the MY_AREA storage area:

```
$ RMU/DUMP/AREA=MY_AREA/START=5/END=10 test_db/OUT=rmu_dump_area.out
```

Search the rmu_dump_area.out file for the occurrence of "total hash bucket" and "static data" as follows:

```
$ SEARCH rmu_dump_area.out "total hash bucket"
                                  ....  total hash bucket size: 97
                                  ....  total hash bucket size: 118
                                  ....  total hash bucket size: 118
                                  ....  total hash bucket size: 118
                                  ....  total hash bucket size: 118
                                  ....  total hash bucket size: 118
                                  ....  total hash bucket size: 118
                                  ....  total hash bucket size: 118
                                  ....  total hash bucket size: 118

   .
   .
   .
$ SEARCH rmu_dump_area.out "static data"
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
                                  ....  311 bytes of static data
   .
   .
   .
```

The hash bucket size is 118 bytes and the data row size is 311 bytes. Other rows in this table may require more or less space. It is important to scan a representative sample of pages randomly to determine the appropriate row size. Oracle Rdb rounds row sizes up to the next longword.

The Performance Monitor row caching screens provide statistics on inserting rows into a cache. One of the statistics, “row too big”, indicates that a row is too large to fit into the specified cache. See Section 4.1.4.6 for more information on the Performance Monitor screens related to row caching.

The slot count multiplied by the slot size specifies the approximate size, in bytes, of the row cache. You should also take into account additional overhead.

4.1.4.1.3 Allocating Memory When you create a row cache or modify a row cache definition, you have the option of specifying where, in memory, you want Oracle Rdb to create the cache. Row caches can reside in the following memory locations:

- Process global section on OpenVMS or shared memory partition on Digital UNIX

  When you use global sections created in the process space, you and other users share physical memory and the OpenVMS operating system maps a cache to a private address space for each user.

  Use the SHARED MEMORY IS PROCESS parameter to specify that the cache be created in a process global section or shared memory partition as shown in the following example:

  ```
  SQL>
  SQL> ALTER DATABASE FILENAME mf_personnel
  cont> ADD CACHE EMPIDS_LOW_RCACHE
  cont> SHARED MEMORY IS PROCESS;
  ```

  This is the default.

OpenVMS Alpha

- System space buffer

  The system space global section is located in the OpenVMS Alpha system space, which means that a system space global section is fully resident, or pinned, in memory and does not affect the quotas of the working set of a process.

  System space is critical to the overall system. System space buffers are not paged; therefore, they use physical memory, reducing the amount of physical memory available for other system tasks. This may be an issue if your system is constrained by memory. You should be careful when you allocate system space. Nonpaged dynamic pool (NPAGEDYN) and the VAXcluster cache (VCC) are some examples of system parameters that use system space.

  Use the SHARED MEMORY IS SYSTEM parameter to specify that the cache be created in a system space buffer, as shown in the following example:

  ```
  SQL>
  SQL> ALTER DATABASE FILENAME 'mf_personnel'
  cont> ADD CACHE EMPIDS_MID_RCACHE
  cont> SHARED MEMORY IS SYSTEM;
  ```

Consider allocating small caches that contain heavily accessed data in system space buffers. When a row cache is stored in a system space buffer, there is no process overhead and data access is very fast because the data does not need to be mapped to user windows. The Hot Row Information screen in the Performance Monitor displays a list of the most frequently accessed rows for a specific row cache. ♦

OpenVMS Alpha

- Very large memory

  Very large memory (VLM) allows Oracle Rdb to use as much physical memory as is available on your system and to dynamically map it to the virtual address space of database users. VLM provides access to a large amount of physical memory through small virtual address windows. Even though VLM is defined in physical memory, the virtual address windows are defined and maintained in each user's private virtual address space.

  Use the LARGE MEMORY parameter to specify that the cache be created in large memory.

  ```
  SQL>
  SQL> ALTER DATABASE FILENAME 'mf_personnel'
  cont> ADD CACHE EMPIDS_OVER_RCACHE
  cont> LARGE MEMORY IS ENABLED;
  SQL>
  ```
  ♦

  VLM is useful for large tables with high access rates. The only limiting factor with VLM is the amount of available physical memory on your system.

You view the physical memory through windows. You can specify the number of window panes with the WINDOW COUNT parameter. By default, Oracle Rdb allocates 100 window panes to a process.

Table 4–5 summarizes the location in memory of each row cache object and whether process private virtual address windows are needed to access the data.

Table 4–5 Memory Locations of Row Cache Objects

SHARED	LARGE	Control Structures	Data Rows	Windows
PROCESS[1]	DISABLED[3]	Process global section or shared memory partition	Process global section or shared memory partition	No
PROCESS[1]	ENABLED[4]	Process global section or shared memory partition	Physical memory	Yes
SYSTEM[2]	DISABLED[3]	System space	System space	No
SYSTEM[2]	ENABLED[4]	System space	Physical memory	Yes

[1] SHARED MEMORY IS PROCESS

- The row cache control structures are located in a process global section or shared memory partition.
- The storage of the data rows depends on whether large memory is enabled or disabled.
  - If large memory is enabled, data is stored in physical memory and windows from each user's process virtual address space are needed to access the data.
  - If large memory is disabled, data is stored in a process global section or shared memory partition and no windows are needed to access the data.

[2] SHARED MEMORY IS SYSTEM

- The row cache control structures are stored in system space.
- The storage of the data rows depends on whether large memory is enabled or disabled.
  - If large memory is enabled, data is stored in physical memory and windows from each user's process virtual address space are needed to access the data.
  - If large memory is disabled, data is stored in system space and no windows are needed to access the data.

[3] LARGE MEMORY IS DISABLED

- The storage of the data rows and the row cache control structures depends on whether shared memory is process or system.
  - If shared memory is process, the data and row cache control structures are stored in a process global section or shared memory partition and no windows are needed to access the data.
  - If shared memory is system, the data and row cache control structures are stored in system space and no windows are needed to access the data.

[4] LARGE MEMORY IS ENABLED

- The data rows are stored in physical memory and process private virtual address windows are needed to access the data.
- The storage of the row cache control structures depends on whether shared memory is process or system.
  - If shared memory is process, the control structures are stored in a process global section or shared memory partition.
  - If shared memory is system, the control structures are stored in system space.

It is important that you consider the amount of memory available on your system before you start creating and using row caches. However, there are currently no tools available to determine the amount of system space available on the system. Because VLM row caches can consume a certain amount of system space for their virtual address windows, Oracle Corporation recommends that you define and activate the VLM caches first, so that any VLM system space requirements are satisfied before you activate system space buffer row caches. After the VLM row caches are defined, you can define system space buffer row caches for small tables that contain frequently accessed data.

OpenVMS VAX OpenVMS Alpha

On OpenVMS systems, you can use the DCL command SHOW MEMORY /PHYSICAL to check the availability and usage of physical memory. This command displays information on how much memory is used and how much is free. The free memory is available for VLM row caches in addition to user applications.

The following example shows a system that has 1.5 gigabytes of memory or a total of 196608 OpenVMS Alpha memory pages (an OpenVMS Alpha page is 8192 bytes):

```
$ SHOW MEMORY/PHYSICAL

              System Memory Resources on 29-MAY-1996 21:39:35.40

Physical Memory Usage (pages):     Total        Free      In Use    Modified
  Main Memory (1536.00Mb)         196608      183605       12657         346
```

Of the 1.5 gigabytes, 183605 pages remain on the free list. Most of this free memory is available for row cache allocation.

Assume a logical area cache has been defined for the MY_TABLE table. The following SQL statement maps the logical area cache:

```
SQL> ATTACH 'FILE test_db';
SQL> SELECT * FROM MY_TABLE WHERE MY_HASH_INDEX = 100;
```

By issuing this SQL statement, the logical area cache has allocated the necessary memory accounting for 40462 OpenVMS Alpha pages, as shown in the following SHOW MEMORY/PHYSICAL command output:

```
$ SHOW MEMORY/PHYSICAL

              System Memory Resources on 29-MAY-1996 21:46:07.01

Physical Memory Usage (pages):     Total        Free      In Use    Modified
  Main Memory (1536.00Mb)         196608      143143       52766         699
```

Notice the drop in the amount of free memory.

The following SHOW MEMORY/PHYSICAL command was issued after users attached to the database, allocated their working sets, and began to work:

```
              System Memory Resources on 29-MAY-1996 23:48:06.67
Physical Memory Usage (pages):     Total        Free      In Use    Modified
Main Memory (1536.00Mb)           196608       81046      112498        3064
```

In this example, only 81046 OpenVMS Alpha pages are left on the free list. ♦

4.1.4.1.4 Assigning Row Caches to Storage Areas A row cache is associated with a particular storage area. One row cache can contain rows from one or more storage areas, but one storage area can point to only one row cache. The following example shows how to assign a row cache to a storage area:

```
SQL> ALTER STORAGE AREA
cont> EMPIDS_LOW CACHE USING EMPIDS_LOW_RCACHE;
```

You can also define a default row cache for all of the storage areas in the database by using the CACHE USING clause of the ALTER DATABASE or CREATE DATABASE statement.

4.1.4.1.5 Assigning Row Caches to Tables A logical area row cache is associated with all partitions of a specific table or index. In the following example, the PARTS table is partitioned across five storage areas:

```
SQL> CREATE STORAGE MAP INVENT_MAP FOR PARTS
cont> ! --
cont> ! -- Inventory table partitioned by stock number
cont> ! --
cont>  STORE USING (STOCK_ID)
cont>      IN STOCKID_A_E WITH LIMIT OF ('10000')
cont>      IN STOCKID_F_K WITH LIMIT OF ('20000')
cont>      IN STOCKID_L_P WITH LIMIT OF ('30000')
cont>      IN STOCKID_Q_U WITH LIMIT OF ('40000')
cont>      OTHERWISE IN STOCKID_V_Z
cont>      PLACEMENT VIA INDEX STOCK_HASH;
SQL>
   .
   .
   .
```

Suppose you want to store all of the PARTS table in a single row cache. Define a row cache with the same name as the table:

```
SQL> ALTER DATABASE FILENAME 'INVENTORY'
cont> ADD CACHE PARTS
cont> ROW LENGTH IS 100 BYTES
cont> CACHE SIZE IS 5000 ROWS;
```

All rows from all partitions of the PARTS table are automatically cached because the cache name is exactly the same as the table name.

4.1.4.2 Controlling What Gets Cached in Memory

The ROW REPLACEMENT parameter gives you some control over what happens when a row cache becomes full. If row replacement is enabled for a particular row cache, then new rows will replace the oldest, unused, unmarked rows once the cache is full. If row replacement is disabled, then new rows are not placed in the cache once the cache is full; they will always be retrieved from disk.

The ROW REPLACEMENT parameter allows you to pin rows in memory. You can increase performance by pinning the following:

- Nonleaf nodes of a B-tree index

  Be sure to account for the nodes splitting when you specify the size for the row cache. If a parent node splits and there is no room in the cache for the new node, the new node will not be pinned in memory.

- Data that is primarily read-only

  Data that is not subject to change very often, such as fact tables in a data warehouse environment, is a good candidate for keeping resident in memory.

- Data that is update-intensive, when the entire table can fit in the cache

  Oracle Rdb optimizes access when the cache is defined to be no replacement.

When you use the ROW REPLACEMENT IS DISABLED clause, the data becomes memory resident and all subsequent reads occur from memory rather than disk. This also improves performance because Oracle Rdb requires less locking.

Enabling row replacement is beneficial when access patterns of a table are random. This ensures that the most frequently accessed rows remain in memory. Often, there may not be enough physical memory to cache an entire table, so caching the most frequently used rows can improve performance.

4.1.4.2.1 Row Replacement Strategy Global and local buffers use the least-recently used (LRU) replacement strategy for database pages. Row caching uses a modified form of the LRU replacement strategy. Each database user can protect the last 10 rows he or she accessed. This group of rows is referred to as a **working set**. Rows that belong to a working set are considered to be **referenced** and are not eligible for row replacement. Any row that is in a cache and is not part of a working set is considered an **unreferenced** row. The unreferenced rows are eligible for replacement.

4.1.4.3 Inserting Rows into a Cache

Each user process requests rows from the database. A user process, which reads a row from a storage area, tries to insert the row into the cache (if it is not already there). If a slot is available, the requested row is stored in the cache, if it fits. If no more slots are available in the cache, one of the following happens:

- If ROW REPLACEMENT IS ENABLED, and an unreferenced row can be found, the unreferenced row is replaced by the new row. Oracle Rdb chooses the unreferenced row randomly.
- If ROW REPLACEMENT IS DISABLED, the row is not stored in the cache.

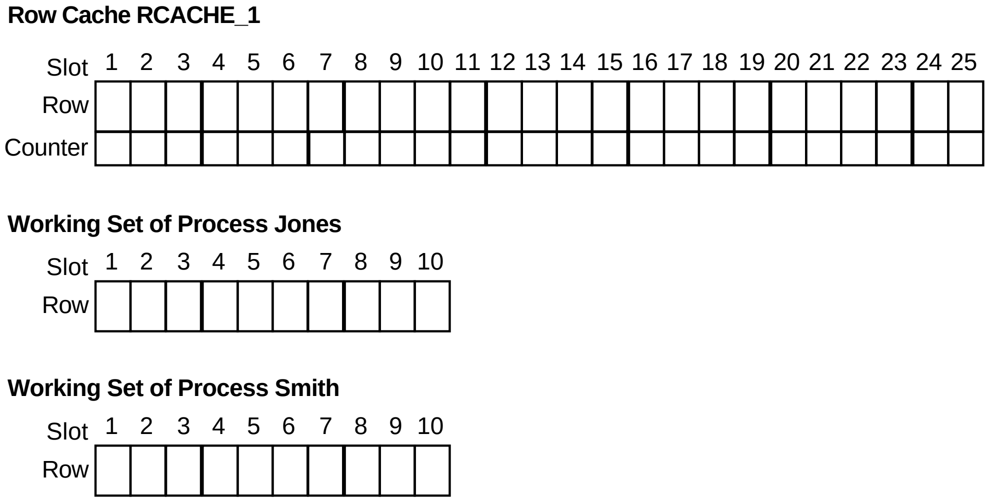
If system usage is lighter at night, you may want to preload row caches at night so that the data is available in memory during the day when database activity is at its peak.

The remainder of this section illustrates how Oracle Rdb inserts rows into a cache.

The example makes the following assumptions:

- Row caching is enabled.
- Row replacement is enabled.
- A row cache (RCACHE_1) has been created with 25 slots.
- Two processes (Jones and Smith) are attached to the database.
- The rows in the row cache are not modified.

The initial allocation is as follows:



Row Cache RCACHE_1

Slot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Row																									
Counter																									

Working Set of Process Jones

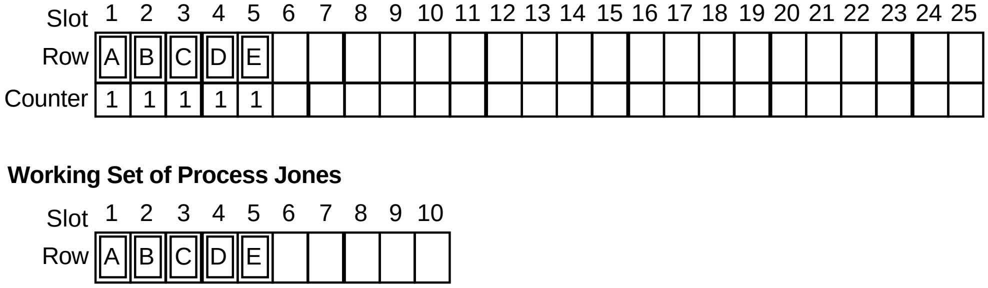
Slot	1	2	3	4	5	6	7	8	9	10
Row										

Working Set of Process Smith

Slot	1	2	3	4	5	6	7	8	9	10
Row										

NU−3614A−RA

1. Process Jones executes a query that causes 5 rows to be read into the first 5 slots of the row cache.



Slot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Row	A	B	C	D	E																				
Counter	1	1	1	1	1																				

Working Set of Process Jones

Slot	1	2	3	4	5	6	7	8	9	10
Row	A	B	C	D	E					

NU−3615A−RA

Each row slot has a working set counter associated with it. The working set counter indicates whether the row belongs to a working set. A positive value indicates that the row belongs to a working set. If a row belongs to a working set, it is not eligible for row replacement.

2. Process Smith requests 15 rows from the database. The first 10 rows requested go into Smith's working set as follows:

Working Set of Process Smith

Slot	1	2	3	4	5	6	7	8	9	10
Row	F	G	H	I	J	K	L	M	N	O

NU−3616A−RA

Process Smith's working set has exactly 10 slots, and all 10 are being used. The least recently used row is replaced by the eleventh row that Process Smith reads into the cache. Rows 12 through 15 also overwrite the contents of slots 2 through 5 respectively.

After the 15 rows are read into the cache, the cache appears as follows:

Slot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Row	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T					
Counter	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1					

NU−3617A−RA

After the 15 rows are read into the cache, Process Smith's working set appears as follows:

Working Set of Process Smith

Slot	1	2	3	4	5	6	7	8	9	10
Row	P	Q	R	S	T	K	L	M	N	O

NU−3618A−RA

At this point, rows F, G, H, I, and J are unreferenced. They are in the cache but they do not belong to the working set of any process. Oracle Rdb sets the working set counter for an unreferenced row to zero. The unreferenced rows are eligible for replacement if they have not been modified and row replacement is enabled. Any process can read rows F, G, H, I, or J without executing an I/O. However, if a process requires a row that is not currently in the cache, one of the rows F, G, H, I, or J is replaced with the new row.

Each slot in the row cache contains a modification flag. If the row has been modified, but not flushed to disk yet, it is considered to be **dirty**. Dirty rows are not candidates for row replacement either. Modified rows are written to disk by the row cache server (RCS) process. See Section 4.1.4.4 for more information.

3. Process Jones requests 7 more rows: M, U, V, W, X, Y, and Z. Jones can read row M without performing any I/O because M is already in the cache. An additional slot does not get filled in the row cache, but row M is added to Process Jones' working set.

   Process Jones' working set now appears as follows:

Working Set of Process Jones

Slot	1	2	3	4	5	6	7	8	9	10
Row	Y	B	C	D	E	M	U	V	W	X

NU−3619A−RA

Rows U, V, W, X, and Y go into the remaining slots in the row cache and the row cache appears as follows:

Slot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Row	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Counter	0	1	1	1	1	0	0	0	0	0	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1

NU−3620A−RA

Note that the working set counter for slot 13 indicates that row M is in two working sets. This indicates that two processes are accessing the same row. The number of processes sharing a particular slot is known as the **share count**.

At this point, the cache is full. If row replacement were disabled for the row cache, then row Z could not be inserted. However, in this example, row replacement is enabled, and there is an unreferenced slot. Therefore, Oracle Rdb will choose a victim from the unreferenced slots to make room for the new row, Z. (In this example, the unreferenced slots are A, F, G, H, I, and J.)

4.1.4.4 Row Cache Server (RCS) Process

When you enable row caching, Oracle Rdb delays writing committed updates to storage area files until a user-specified threshold, called a **sweep threshold**, is reached. All modified rows since the last sweep are written to their respective storage areas. Sweeping of the rows in the row cache is done by the **row cache server (RCS)** process.

The RCS process performs the following functions:

- Writes marked cold records from row caches to cache backing store (.rdc) files

The RCS process writes to the cache backing store files when a checkpoint interval is reached.

- Writes marked cold records from row caches to storage area (.rda) files

  The RCS process writes to the storage area files when it sweeps the row caches.

Oracle Rdb associates a modification flag for every row in a cache to indicate which rows have been modified. The modification flag can accept the following values:

Modification Flag	Meaning
Marked Cold (MC)	The row has been modified and is available to be written to the .rda file during the next sweep.
Marked Hot (MH)	The row has been modified. The row's modification flag will change to marked cold during the next sweep.
Unmarked Cold (UC)	The row has not been modified.

The RCS is a detached server process. Oracle Rdb creates the RCS process when row caching is enabled for the database.

4.1.4.4.1 Row Cache Checkpointing

Oracle Rdb creates a cache backing store file with an .rdc file extension for each row cache. The RCS process writes to these files when the checkpoint interval is reached.

You can specify a checkpoint interval by using the following SQL syntax:

- CHECKPOINT INTERVAL IS n BLOCKS

  Specifies a checkpoint interval that indicates the number of blocks the .aij file is allowed to accumulate before the modified rows are written to the cache backing store files.

- CHECKPOINT TIMED EVERY s SECONDS

  Specifies a checkpoint interval that indicates the number of seconds that can pass before the modified rows are written to the cache backing store files.

You can use the RDM$BIND_RCS_CHECKPOINT logical name or the RDB_BIND_RCS_CHECKPOINT configuration parameter to indicate whether the RCS process performs a checkpoint. The value 1 indicates a checkpoint is performed, and the value 0 indicates a checkpoint is not performed.

By default, the .rdc files are 40 percent of the row cache size. If you have a read-only cache, you should specify 1 block for the size of the .rdc file as follows:

```
SQL> ALTER DATABASE FILE mf_personnel
cont> ADD CACHE RCACHE_2
cont> LOCATION IS WORK$DISK1:[RCS]
cont> ALLOCATION IS 1 BLOCK;
```

If you do not specify a location for the .rdc files, the default location is the database root file directory.

4.1.4.4.2 Row Cache Sweeping One purpose of the RCS process is to sweep the row caches. A **sweep** is one full pass through all active row caches, attempting to write modified rows back to their respective storage areas. Ideally, the RCS should write rows to storage areas before a cache becomes full. A database administrator (DBA) can specify the following:

- How full the cache should get before modified rows are written to the storage area files

  This is accomplished by setting a sweep threshold.

- How often the cache should be swept

  This is accomplished by setting a sweep interval.

When the RCS process is sweeping and finds a row that has been modified, the following occurs:

- If the modified row is marked cold, the modified row is written to the storage area file and the modification flag is changed to unmarked cold.

  When the modification flag is unmarked cold, the slot becomes a candidate for row replacement.

- If the modified row is marked hot, the modification flag is changed to marked cold. It will be written to the storage area file during the next sweep.

- Flush counter is incremented by one

  Oracle Rdb keeps track of the number of times a row in a cache has been written to the storage area file.

The RCS process sweeps the row caches sequentially in the order in which they were created.

Every row cache has two thresholds associated with it: a “start of sweep” threshold and an “end of sweep” threshold. When the number of modified rows exceeds the “start of sweep” threshold, then a sweep is started until the number of modified rows goes below the “end of sweep” threshold.

You can specify the sweep thresholds with the following logical names and configuration parameters:

- RDM$BIND_RCS_MAX_COLD and RDB_BIND_RCS_MAX_COLD

  This defines the “start of sweep” threshold.

- RDM$BIND_RCS_MIN_COLD and RDB_BIND_RCS_MIN_COLD

  This defines the “end of sweep” threshold.

OpenVMS VAX ≡ OpenVMS Alpha ≡

The following example shows how to set the sweep thresholds to 20 and 50 percent respectively:

```
$ DEFINE/SYSTEM RDM$BIND_RCS_MIN_COLD 20
$ DEFINE/SYSTEM RDM$BIND_RCS_MAX_COLD 50
$!
$! When the number of modified rows in a row cache exceeds
$! 50%, Oracle Rdb writes the modified rows to the
$! storage area.
$!
$! This sweeping of modified rows occurs until the number of
$! modified rows goes below 20%.
$!
```

You must define these as system level logical names. ♦

You can specify a sweep interval for a row cache. The sweep interval indicates the amount of time, in minutes, between each RCS sweep. Use the RDM$BIND_RCS_SWEEP_INTERVAL logical name or the RDB_BIND_RCS_SWEEP_INTERVAL configuration parameter to define the sweep interval.

OpenVMS VAX ≡ OpenVMS Alpha ≡

The following example shows how to set the sweep interval to two minutes:

```
$ DEFINE/SYSTEM RDM$BIND_RCS_SWEEP_INTERVAL 2
$!
$! The RCS process takes 2 minutes to write the modified
$! rows to the storage areas. The sweep interval allows
$! the DBA to balance the activity of the RCS process with
$! other users accessing the database.
$!
```

You must define RDM$BIND_RCS_SWEEP_INTERVAL as a system level logical name. ♦

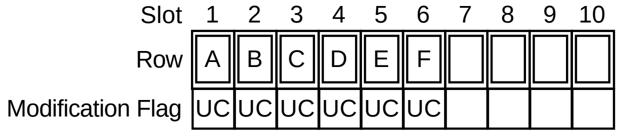
You can indicate that the RCS process initiate a sweep even when a threshold has not been reached. Invoke the tools facility within the Performance Monitor by entering the exclamation point (!), and select the Wake up RCS process option.

If the sweep thresholds or sweep intervals are not set properly, it is possible for a row cache to become full of modified rows, and Oracle Rdb will not be able to insert a new row into the cache until the RCS process writes the modified rows to the storage area files and clears the modification flag. You can determine if caches are filling up with modified rows by looking at the "cache full" statistic on the Row Cache (One Cache) screen in the Performance Monitor.

Row caches that are frequently filled with modified rows are an indication that you should adjust the sweep thresholds and the sweep interval. You can experiment with different sweep thresholds and sweep intervals by using the RCS Dashboard in the Performance Monitor. When you determine acceptable settings, make the settings persistent by defining the appropriate logical names or configuration parameters.

The remainder of this section shows what happens when the RCS process performs a sweep.

1. Process 1 executes a query that causes 6 rows to be stored in the cache:

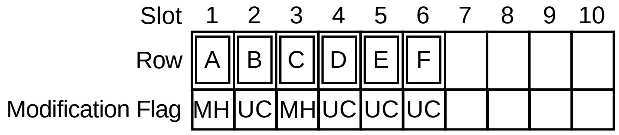


Slot	1	2	3	4	5	6	7	8	9	10
Row	A	B	C	D	E	F				
Modification Flag	UC	UC	UC	UC	UC	UC				

NU−3632A−RA

   At this point, all the records in the cache are considered to be "unmarked cold" because none of them have been modified yet.

2. Process 2 updates the rows in slots 1 and 3.



Slot	1	2	3	4	5	6	7	8	9	10
Row	A	B	C	D	E	F				
Modification Flag	MH	UC	MH	UC	UC	UC				

NU−3633A−RA

   Slots 1 and 3 are now "marked hot".

3. The RCS process initiates a sweep doing the following:

   - For any "marked cold" slots, RCS writes the rows to the storage area and changes the modification flag from "marked cold" to "unmarked cold."

- For any “marked hot” slots, RCS changes them to “marked cold.”

Slot	1	2	3	4	5	6	7	8	9	10
Row	A	B	C	D	E	F				
Modification Flag	MC	UC	MC	UC	UC	UC				

NU−3634A−RA

The first time the RCS process sweeps the row cache, it does not write any modified rows to the storage area. If a row has been modified, the RCS changes the modification flag from “unmarked cold” to “marked hot.”

4. Process 2 executes a query that causes rows in slots 5 and 6 to be updated.

Slot	1	2	3	4	5	6	7	8	9	10
Row	A	B	C	D	E	F				
Modification Flag	MC	UC	MC	UC	MH	MH				

NU−3635A−RA

5. The RCS process comes along and sweeps, doing the following:

   - For any “marked cold” slots, RCS writes the rows to the storage area and changes the modification flag from “marked cold” to “unmarked cold.”
   - For any slots that are “marked hot,” the RCS process changes them to “marked cold.”

   After the sweep, the cache looks as follows:

Slot	1	2	3	4	5	6	7	8	9	10
Row	A	B	C	D	E	F				
Modification Flag	UC	UC	UC	UC	MC	MC				

NU−3636A−RA

During this sweep, the rows in slots 1 and 3 are written to the storage area file, and the modification flags for slots 1 and 3 are changed from “marked cold” to “unmarked cold.” Note that the RCS process will write the rows in slots 5 and 6 to the storage area during the next sweep.

4.1.4.5 Using Physical and Logical Area Caches

You can have both physical and logical area caches. For example, assume the following sizes for the records in a hashed index:

- system record—16 bytes (STOCK_SYS cache)
- hash bucket record —100 bytes (STOCK_HASH cache)
- data record —320 bytes (STOCK cache)

If you created one cache for all three record types, with a row size of 320 bytes, you would waste a lot of memory when storing the system record and the hash bucket record. It would be more efficient to have three caches, one for each record type.

Example 4–12 shows the three row cache definitions.

Example 4–12 Row Cache Definition

```
SQL> --
SQL> -- The following cache definition is for the system records:
SQL> --
cont>  ADD CACHE STOCK_SYS
cont>      CACHE SIZE IS 5000 rows
cont>      ROW LENGTH IS 16 bytes
cont>      NUMBER OF RESERVED ROWS IS 1200
cont>      SHARED MEMORY IS SYSTEM
cont>      LARGE MEMORY IS ENABLED
cont>      ROW REPLACEMENT IS ENABLED
cont>      LOCATION IS '$1$DKA400:[RCS]'
cont>      ALLOCATION IS 1 BLOCK
cont>      EXTENT IS 1 BLOCK
cont>      NUMBER OF SWEEP ROWS IS 1000;
```

(continued on next page)

Example 4–12 (Cont.) Row Cache Definition

```
SQL> --
SQL> -- The following cache definition is for the hash bucket:
SQL> --
cont>  ADD CACHE STOCK_HASH
cont>      CACHE SIZE IS 5000 rows
cont>      ROW LENGTH IS 100 BYTES
cont>      NUMBER OF RESERVED ROWS IS 1200
cont>      SHARED MEMORY IS SYSTEM
cont>      LARGE MEMORY IS ENABLED
cont>      ROW REPLACEMENT IS ENABLED
cont>      LOCATION IS '$1$DKA400:[RCS]'
cont>      ALLOCATION IS 1 BLOCK
cont>      EXTENT IS 1 BLOCK
cont>      NUMBER OF SWEEP ROWS IS 1000;
SQL>
SQL> --
SQL> -- The following cache definition is for the data records:
SQL> --
cont>  ADD CACHE STOCK
cont>      CACHE SIZE IS 5000 rows
cont>      ROW LENGTH IS 320 BYTES
cont>      NUMBER OF RESERVED ROWS IS 2400
cont>      SHARED MEMORY IS SYSTEM
cont>      LARGE MEMORY IS ENABLED
cont>      ROW REPLACEMENT IS ENABLED
cont>      LOCATION IS '$1$DKA400:[RCS]'
cont>      ALLOCATION IS 1750000 BLOCKS
cont>      EXTENT IS 1500000 BLOCK
cont>      NUMBER OF SWEEP ROWS IS 1000;
SQL>
```

If one cache was used for all three record types, the row size would have to be large enough to accommodate the largest row. In Example 4–12, the data row is 320 bytes. If you add 10 bytes for overhead, the amount of memory utilized with one row cache would be as follows:

```
Total
number    = (# of rows in cache * row length of largest row)
of bytes

          = (15000 * 330)

          = 4950000 bytes
```

The amount of memory utilized with 3 row caches is computed as follows:

```
Total
number    = (# of rows in cache * row length of system record) +
of bytes    (# of rows in cache * row length of hash bucket) +
            (# of rows in cache * row length of data record)

          = (5000 * 16) +
            (5000 * 100) +
            (5000 * 330)

          = 1735000 bytes
```

There is a considerable savings in memory.

4.1.4.6 Performance Monitor Screens and Row Caching

The Performance Monitor provides the following screens you can use to display statistics about row caching performance and effectiveness.

- Summary Cache Statistics
- Summary Cache Unmark Statistics
- Row Cache (One Cache)
- Row Cache (One Field)
- Row Cache Utilization
- Hot Row Information
- Row Cache Status
- Row Cache Queue Length
- Row Length Distribution
- RCS Statistics
- Row Cache Dashboard
- RCS Dashboard
- Per-Process Row Cache Dashboard

For detailed information on each of these screens, see the Performance Monitor help. The remainder of this section shows some examples of the row caching screens.

4.1.4.6.1 Row Cache (One Cache)

The following example shows that the row length was not sized properly, and as a result, none of the rows fit into the cache:

```
Node: BONZAI         Oracle Rdb V7.0-00 Performance Monitor 11-APR-1996 16:24:57
Rate: 3.00 Seconds           Row Cache (EMPIDS_LOW)         Elapsed: 00:02:54.98
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------
statistic........... rate.per.second............. total....... average......
name................ max..... cur..... avg....... count....... per.trans....
latch requests              0        0        0.0            0           0.0
  retried                   0        0        0.0            0           0.0
cache searches            536      530      507.5        27066       27066.0
  found in workset          0        0        0.0            0           0.0
  found in cache            0        0        0.0            0           0.0
  found too big             0        0        0.0            0           0.0
insert cache              536      530      507.5        27066       27066.0
  row too big             536      530      507.5        27066       27066.0
  cache full                0        0        0.0            0           0.0
  collision                 0        0        0.0            0           0.0
VLM requests                0        0        0.0            0           0.0
  window turns              0        0        0.0            0           0.0
skipped dirty slot          0        0        0.0            0           0.0
skipped inuse slot          0        0        0.0            0           0.0
hash misses                 0        0        0.0            0           0.0
cache unmark                0        0        0.0            0           0.0
--------------------------------------------------------------------------------
Exit Graph Help Menu Options Reset Set_rate Time_plot Unreset Write X_plot !
```

The following example shows that every requested row was found in the cache:

```
Node: BONZAI         Oracle Rdb V7.0-00 Performance Monitor 11-APR-1996 16:28:03
Rate: 3.00 Seconds           Row Cache (EMPIDS_MID)         Elapsed: 00:02:58.81
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------
statistic........... rate.per.second............. total....... average......
name................ max..... cur..... avg....... count....... per.trans....
latch requests              0        0        0.3         4763         297.6
  retried                   0        0        0.0            0           0.0
cache searches           2585       95       10.3       151297        9456.0
  found in workset          0        0        0.0            0           0.0
  found in cache         2585       95        3.4        51297        3206.0
  found too big             0        0        0.0            0           0.0
insert cache                6        0        6.8       100000        6250.0
  row too big               0        0        0.0            0           0.0
  cache full                0        0        0.0            0           0.0
  collision                 0        0        0.0            0           0.0
VLM requests                0        0        0.0            0           0.0
  window turns              0        0        0.0            0           0.0
skipped dirty slot          0        0        0.0            0           0.0
skipped inuse slot          0        0        0.0            0           0.0
hash misses               820        0        2.4        36237        2264.8
cache unmark                0        0        0.0            0           0.0
--------------------------------------------------------------------------------
Exit Graph Help Menu Options Reset Set_rate Time_plot Unreset Write X_plot !
```

The "found in workset" and "found in cache" statistics indicate how efficient the cache is. The closer these statistics are to 100%, the better the row cache utilization.

The cache hit rate in the following example is around 75%:

```
Node: BONZAI         Oracle Rdb V7.0-00 Performance Monitor 11-APR-1996 16:33:03
Rate: 3.00 Seconds          Row Cache (EMPIDS_MID)         Elapsed: 00:03:03.10
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------
statistic.......... rate.per.second............ total....... average......
name............... max..... cur..... avg....... count....... per.trans....
latch requests              18       16       0.3       11143          0.3
  retried                    0        0       0.0           0          0.0
cache searches            2914     1958      51.4     1501158         50.0
  found in workset         355      225       5.2      154416          5.1
  found in cache          1851      883      19.3      564795         18.8
  found too big              0        0       0.0           0          0.0
insert cache               721      608      14.6      429294         14.3
  row too big                0        0       0.0           0          0.0
  cache full                58       58       0.7       22839          0.7
  collision                  0        0       0.0           3          0.0
VLM requests              1957     1041      18.2      533613         17.7
  window turns             117       33       1.1       34495          1.1
skipped dirty slot           0        0       0.0           0          0.0
skipped inuse slot           3        0       3.8      112321          3.7
hash misses                391      391       5.5      162170          5.4
cache unmark               235      116       3.1       91713          3.0
--------------------------------------------------------------------------------
Exit Graph Help Menu Options Reset Set_rate Time_plot Unreset Write X_plot !
```

The following example shows an RCS Dashboard screen:

```
Node: BONZAI         Oracle Rdb V7.0-00 Performance Monitor 11-APR-1996 16:34:00
Rate: 3.00 Seconds              RCS Dashboard              Elapsed: 00:11:57.85
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------


Database.......... Current... Previous.. Lowest.... Highest... Original.. Chng
Attribute.Name.... Value..... Value..... Value..... Value..... Value..... Cnt.

Ckpt Buffer Count         15         15         15         15         15    0
Batch Count            10000      10000      10000      10000      10000    0
Checkpoint                 1          1          1          1          1    0
Ckpt Time Interval       300        300        300        300        300    0

Sweep Interval             1          1          1          1          1    0
Low Cold Thshld      2899000    2899000    2899000    2899000    2899000    0
High Cold Thshld     2900000    2900000    2900000    2900000    2900000    0
Cold Record Count          0          0          0          0          0    0
Abort Sweep                1          1          1          1          1    0


--------------------------------------------------------------------------------
Config Exit Help Menu Options Set_rate Write !
```

The row caching screens do not appear in the Performance Monitor main menu if one or more of the requirements described in Section 4.1.3.1 have not been met.

4.1.5 Fast Commit Transaction Processing

You can improve database performance in some environments if you specify how and when Oracle Rdb writes updated pages from memory buffers to disk.

- By default, Oracle Rdb writes updated database pages to disk each time a transaction executes the COMMIT statement. If a transaction fails before committing, Oracle Rdb only needs to roll back the current failed transaction; it never has to reprocess previous successful transactions.
- By enabling **fast commit** transaction processing, (see Section 4.1.5.4) Oracle Rdb keeps updated pages in a buffer pool and does not write the pages to disk when a transaction commits. The updated pages can remain in the buffer pool until a user-specified threshold (called a checkpoint; see Section 4.1.5.2) is reached, at which time all the updated pages for multiple transactions are flushed to disk. If a transaction fails, Oracle Rdb must roll back the current, failed transaction and reprocess all the committed transactions since the last checkpoint.

Fast commit transaction processing applies only to data updates; that is, to delete, update, and insert operations. Transactions that include data definition statements, such as create logical area or create index operations, force a checkpoint at the end of the transaction.

Note

You must have after-image journaling (AIJ) enabled to use fast commit processing.

The remainder of this section describes default commit processing; Section 4.1.5.1 describes fast commit processing.

With the default method of commit processing, Oracle Rdb executes the following steps for an update transaction (referred to as T1):

1. Starts transaction T1.
2. Reads database pages into the user's buffer.
3. Updates pages.

   When an application or user issues the COMMIT statement:
4. Flushes RUJ buffers to the .ruj file.

5. Flushes updated database pages to disk.
6. Flushes AIJ buffers to the .aij file (if after-image journaling is enabled).
7. Writes commit information for T1 to the database root file.
8. Starts the next transaction, T2.

If transaction T2 fails, the previous, committed transaction T1 is not involved in the recovery, because its updated pages and its commit information have already been written to disk. Oracle Rdb must roll back the failed transaction T2 with information in the .ruj file.

With default commit processing, recovery is very fast because only the effects of *failed* transactions must be undone. This approach optimizes recovery time at the expense of commit time I/O (flushing RUJ buffers to the .ruj file, flushing updated pages to disk, and writing to the database root file).

You should consider using this default method of transaction processing when some or all of the following conditions exist in your environment:

- Your system is subject to system failures or abnormally terminated processes (for example, users pressing Ctrl/Y). The default commit processing method optimizes recovery time.
- The transaction pattern includes:
  - Transactions that update a large number of pages causing buffer pool overflow
  - Multiple update transactions that share a lot of pages, which result in pages being swapped between transactions
  - Transactions that do not repeatedly update the same pages (either within a single transaction, or between transactions)

  With the first two patterns, updated pages must be written to disk regardless of whether fast commit processing is enabled or not. In the last pattern, because a page is updated only once, there is no reason to keep it in the buffer pool.

  Note that observance of these patterns does not mean the default method of commit processing must be used. There are no hard rules; you should make changes using these guidelines, but monitor before and after behavior to determine optimum database performance.

4.1.5.1 Fast Commit Processing Method

When fast commit processing is enabled, Oracle Rdb executes the following steps for an update transaction (referred to as T1):

1. Starts a transaction T1.
2. Reads database pages into the user's buffer.
3. Updates pages, but when an application or user issues the COMMIT statement, does *not* flush the RUJ buffer to the .ruj file, or flush updated database pages to disk.
4. Flushes AIJ buffers to .aij file. After-image journaling *must* be enabled.
5. Writes commit information for T1 to the database root file. This step can also be eliminated if you enable the JOURNAL OPTIMIZATION option when you enable fast commit. See Section 4.1.5.3 for details.
6. Starts the next transaction, T2.
7. Flushes updated pages when the checkpoint is reached, the process detaches from the database, or metadata changes are performed.

If a subsequent transaction fails, all the previous transactions back to the last checkpoint must be redone because their updated pages have not been written to disk. Oracle Rdb uses information written to the .aij file to reprocess these transactions. Oracle Rdb must also roll back the failed transaction with the information in the RUJ buffer or file if changes made by the transaction have been written to the database file (due to buffer pool overflow).

Fast commit processing reduces transaction commit time but increases recovery processing time; if multiple update transactions have committed since the last checkpoint and a failure occurs, all those transactions must be reprocessed. However, you can reduce potential recovery time by setting a shorter checkpoint interval (see Section 4.1.5.2).

You should consider enabling fast commit transaction processing when you have a stable transaction processing environment and the following conditions exist:

- A process updates the same rows for multiple transactions.

  For example, consider a parts database accessed by a user who repeatedly adjusts the quantity on hand as orders are filled and inventory replenished. With default commit processing, pages containing updated rows are flushed to disk at the end of every transaction. This results in excessive disk writes. With fast commit processing enabled, the pages containing the updated rows remain in the buffer pool of the process and are written to disk only at the checkpoint. Not only do you save page I/Os by keeping the updated pages in memory, you also save RUJ I/Os because the RUJ buffer does not get flushed to the .ruj file after each transaction.

- Transactions are short and do not update many pages.

  Under these conditions, it is more efficient to keep the updated pages in the buffer pool and flush them all at the checkpoint rather than flush them at the end of each transaction. For example, issuing ten data I/Os at the same time (at a checkpoint) provides better performance than issuing those same ten I/Os separately (flushing at every commit). In addition you save RUJ I/O operations.

Table 4–6 summarizes the guidelines for using fast commit transaction processing.

Table 4–6 When to Enable or Disable Fast Commit Processing

Transaction Type	Stable Environment	Unstable Environment
Repeated updates to the same rows	Enable	Maybe[1]
Short transactions, few pages updated	Enable	Maybe[1]
Many pages updated with buffer pool overflow	Disable	Disable
Multiple transactions sharing the same pages	Disable	Disable
Infrequent updates to the same rows	Disable	Disable

[1] Enabling fast commit in an environment subject to abnormal failures results in more recovery processing time than the default commit method. Examine the frequency of failure and how much recovery time your system can accommodate when you consider fast commit processing.

The stability of your environment also influences whether or not you should enable snapshot writing if you have fast commit enabled. If your database does not have to recover from frequent system or process failures, you can enable snapshots as either immediate or deferred. If your database does require frequent recoveries, you may want to disable snapshot writing if fast commit is enabled because, for each committed transaction since the last checkpoint, the recovery process must write a record to the snapshot file as well as redo each transaction. Thus recovery time is slower with snapshots enabled. However, if you want to enable the journal optimization option, snapshots must be disabled or deferred.

4.1.5.2 Checkpointing

When you enable fast commit processing, a process does not flush updated pages each time a transaction commits. If the process fails (meaning that changes to memory have been lost), the database recovery (DBR) process must reprocess the process' previous update transactions and roll back any changes made by the aborted transaction. Reprocessing updates involves reading the .aij file and reapplying the changes to the relevant data pages.

You can limit how many transactions the DBR must redo by setting a checkpoint interval. Setting a checkpoint interval instructs Oracle Rdb to periodically flush updated pages. This limits the number of transactions the DBR must reprocess, thereby shortening recovery time. For example, if no checkpoint interval is established and a process completes 1000 transactions but fails during number 1001, the DBR must reprocess transactions 1 through 1000 and roll back number 1001.

Note

When the DBR must perform a recovery, the DBR acquires a freeze lock on the entire database to deny other processes database access. This happens even with default commit processing. However, if fast commit is enabled, the DBR generally has more work to do because multiple transactions may have to be redone. Consequently, the DBR holds the freeze lock longer when fast commit processing is enabled. The following list shows the extra steps required by fast commit processing:

Read .aij file since last checkpoint.	Fast commit
Redo committed transactions not flushed to disk.	Fast commit
Read .ruj file.	Fast commit and default commit
Roll back changes according to RUJ information.	Fast commit and default commit

Fast commit transaction processing has important implications for the AIJ backup procedure. See the *Oracle Rdb7 Guide to Database Maintenance* for information.

When you enable fast commit, you can specify a database-wide checkpoint interval using either or both of the following methods:

- Assign a value that specifies the number of blocks the .aij file is allowed to accumulate before updated pages are flushed. All processes contribute to AIJ growth.

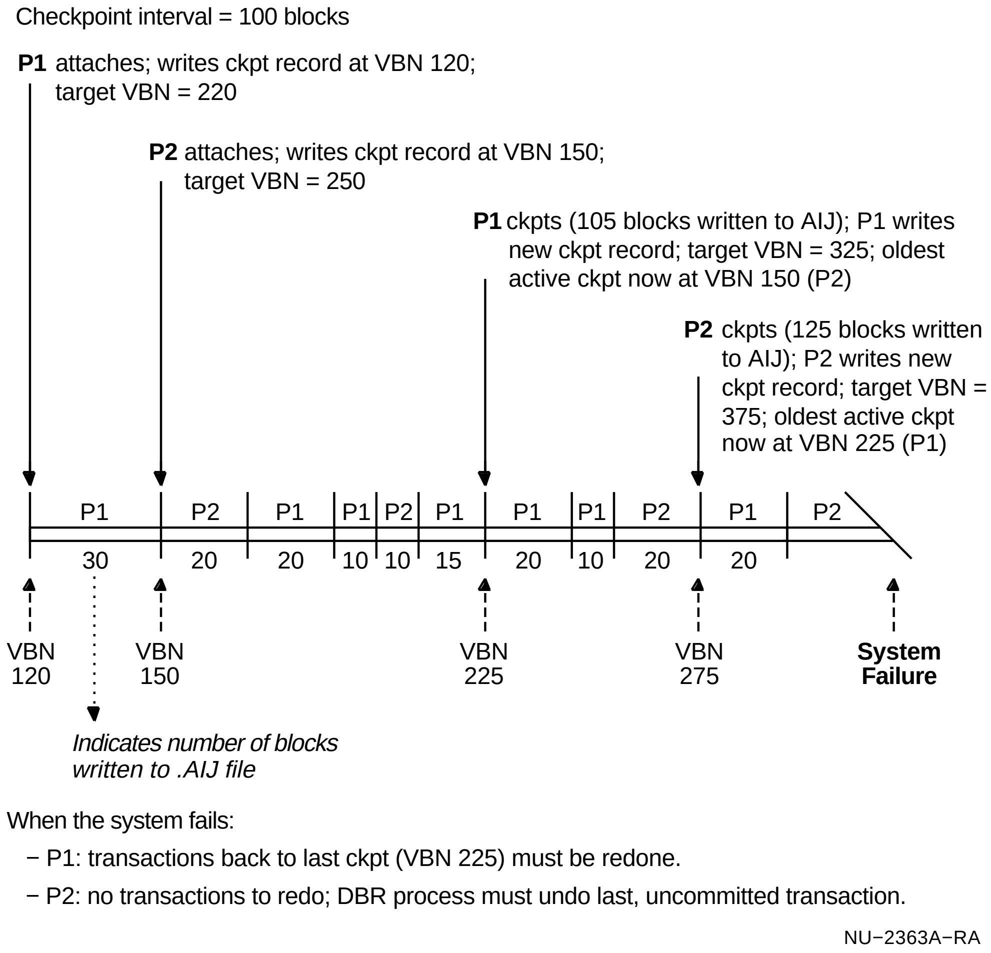
- Assign a value that specifies the elapsed time between checkpoints.

For example, if you set the checkpoint interval value equal to 100 blocks, all processes flush updated pages to disk when 100 blocks have been written to the .aij file since the last checkpoint. Or, if you set the checkpoint interval value equal to 90 seconds, all processes flush updated pages to disk when 90 seconds have elapsed since the last checkpoint. Section 4.1.5.4 describes the fast commit syntax in more detail.

When a process attaches to the database, it writes a checkpoint record to the .aij file and notes the virtual block number (VBN) of the .aij file at which the checkpoint record is located. If the checkpoint is located at VBN 120 and the checkpoint interval is 100 blocks, the process will checkpoint again when VBN 220 is reached. See Figure 4–9. Note that because all processes contribute to .aij file growth, a process may be able to commit many transactions before checkpointing if update activity by other processes is low. Conversely, if a process' first transaction is long and if update activity by other processes is high, the process may be forced to checkpoint when it commits its first transaction. To sum up:

- A checkpoint interval is the number of blocks the .aij file is allowed to increase by or the amount of elapsed time before pages modified by a process are flushed to disk.
- A checkpoint interval applies to each process attached to the database.
- All processes contribute to .aij file growth.
- Checkpointing (updates flushed to disk) occurs at transaction boundaries; a process never checkpoints in the middle of a transaction.

Figure 4–9 Checkpoint Processing


Checkpoint interval = 100 blocks
P1 attaches; writes ckpt record at VBN 120; target VBN = 220
P2 attaches; writes ckpt record at VBN 150; target VBN = 250
P1 ckpts (105 blocks written to AIJ); P1 writes new ckpt record; target VBN = 325; oldest active ckpt now at VBN 150 (P2)
P2 ckpts (125 blocks written to AIJ); P2 writes new ckpt record; target VBN = 375; oldest active ckpt now at VBN 225 (P1)
P1
P2
P1
P1
P2
P1
P1
P1
P2
P1
P2
30
20
20
10
10
15
20
10
20
20
VBN 120
VBN 150
VBN 225
VBN 275
System Failure
Indicates number of blocks written to .AIJ file
When the system fails:
− P1: transactions back to last ckpt (VBN 225) must be redone.
− P2: no transactions to redo; DBR process must undo last, uncommitted transaction.
NU−2363A−RA

When the database checkpoint interval value is reached, Oracle Rdb executes the following steps:

1. Updated pages are flushed to disk.
2. A checkpoint record is written to the .aij file.
3. The database root file is updated for each process to indicate where the checkpoint record is stored in the .aij file. The DBR uses checkpoint information in the root file to determine where in the .aij file recovery must begin.

Consider the following points when you choose a checkpointing interval for your database:

- If the interval is set too small, Oracle Rdb will flush updates too soon and you will not realize the benefits of enabling fast commit processing.
- The larger the interval, the longer the recovery time required to redo the updates of a failed process.
- If you use the elapsed time checkpoint interval, you should know the transaction rate and the number of blocks updated for each transaction on your database. Otherwise, unlike the interval specifying .aij file growth, you will not know how many blocks in the .aij file will need to be recovered after a failure, and recovery time will be unknown.

You must decide how large an .aij file you are willing to let the DBR process in the event of a failure. By inspecting the monitor log file, you can estimate the total recovery time for a given AIJ checkpoint interval. Look for the two log file entries that note when recovery started and when it completed for a process. The difference between the two times is the estimated recovery time. You can use this information to tune the AIJ checkpoint interval. If you want faster recovery, reduce the interval; if you can afford a longer recovery time, increase the checkpoint interval.

To understand how a database administrator (DBA) might decide which checkpointing options to implement, consider the example of a company that takes telephone orders 24 hours a day. Operators, or database users, enter information into the company's database as they take orders.

During the day, the phones are busy. As orders are taken, the .aij file grows rapidly. The DBA knows that:

- One hundred operators work the day shift.
- Each operator takes an average of 10 orders per hour.
- Each order (equivalent to a database transaction) takes 6 minutes to enter.
- Each transaction contributes 1 block to the .aij file.
- The total system generates 1000 transactions per hour.
- The .aij file increases 1000 blocks per hour.

Because the .aij file is growing fairly rapidly at 1000 blocks per hour during the day, the DBA does not want to let the .aij file grow too large before updated pages are written to disk. Remember that if a process fails, DBR has to redo all transactions for that process since it last checkpointed. The more transactions that have occurred since the last checkpoint, the longer recovery will take. If the checkpoint interval is set too small, Oracle Rdb writes updates too soon and you will not realize all of the benefits of enabling fast commit. The larger the interval, the longer the recovery time required to redo the updates of a failed process.

The DBA decides to set a checkpoint interval of 1000 blocks, causing each operator to checkpoint about once per hour.

At night, however, system usage is much lighter. The DBA knows that:

- Two operators work the night shift.
- Each operator takes an average of 10 orders per hour.
- Each order (equivalent to a database transaction) takes 6 minutes to enter.
- Each transaction writes 1 block to the .aij file.
- The system generates only 20 transactions per hour at night, compared to 1000 per hour during the day.
- The .aij file only increases 20 blocks per hour at night.

The DBA realizes that because the .aij file grows so slowly, it will never reach the 1000-block checkpoint interval during the night shift. Of course, if a failure occurs, recovery time will be short because the .aij file is small. The DBA decides that to be safe, updated pages should be written to disk at least a few times during the night. However, the DBA does not want to change the AIJ block interval option, because it works well for the daytime hours. The DBA decides to use the time interval option to force a checkpoint every 2 hours.

Setting a time interval checkpoint does not change the block interval checkpoint, but does force an additional checkpoint every 2 hours for each process to cover the night shift.

In general, AIJ block size is the best indicator of how long recovery will take, and is, therefore, the most meaningful checkpointing option. However, using the time interval option can be helpful if the DBA is unfamiliar with patterns of system usage or the impact of AIJ block size on recovery time.

The checkpoint interval value is set by the DBA and applies to all processes attached to a database. Users can implement an alternate, process-specific method of checkpointing by defining the logical name RDM$BIND_CKPT_TRANS_INTERVAL or the configuration parameter RDB_BIND_CKPT_TRANS_INTERVAL. This mechanism uses transaction count as the checkpoint. For example, if a user defines RDM$BIND_CKPT_TRANS_INTERVAL or RDB_BIND_CKPT_TRANS_INTERVAL to be 10, updated pages are flushed to disk after every tenth transaction *if* the transaction count limit is reached before the checkpoint interval value is reached. Checkpointing is activated by whichever of the three conditions is reached first (.aij file growth, elapsed time, or number of transactions). When a checkpoint occurs, each condition is reset. If fast commit processing is disabled, the RDM$BIND_CKPT_TRANS_INTERVAL logical name and the RDB_BIND_CKPT_TRANS_INTERVAL configuration parameter are ignored.

A user who wants to define RDM$BIND_CKPT_TRANS_INTERVAL or RDB_BIND_CKPT_TRANS_INTERVAL should be very familiar with transaction patterns. For example, if an operator knows that it takes five transactions to process an order, and that some data is modified in each transaction, but that the next order will rarely update data from the previous order, RDM$BIND_CKPT_TRANS_INTERVAL or RDB_BIND_CKPT_TRANS_INTERVAL could be set to 5.

If you want to force all the active processes for a database on all nodes to immediately perform a checkpoint operation, you can do so with the RMU Checkpoint command, as shown in Example 4–13.

Example 4–13 Forcing All Active Database Processes to Perform Immediate Checkpoint Operations

```
$ RMU/CHECKPOINT mf_personnel
$ rmu -checkpoint mf_personnel
```

This command causes all of the modified database cache buffers to be flushed to the disk. The checkpoint operation also improves the redo performance of the DBR, although the per-process parameters should have already been properly initialized with this goal in mind. When the command completes, all active database processes have successfully performed a checkpoint operation. See the *Oracle RMU Reference Manual* for more information on the RMU Checkpoint command.

4.1.5.3 Journal Optimization Option

When you enable fast commit transaction processing, you can enable journal optimization. This option can provide a significant increase in commit processing speed by eliminating the majority of I/O to the database root. This option enhances performance in database environments that are update-intensive. Because of the prerequisites for enabling the journal optimization option (described later in this section), general use databases or databases that have many read-only transactions may not benefit from this feature.

By default, committing a single transaction includes writing to the .ruj file, writing updated pages to disk, writing to the .aij file, and writing commit information to the root file. Writing to the root file becomes a bottleneck for high performance, high transaction throughput applications by restricting the maximum transactions-per-second that an application can achieve.

Enabling fast commit recovery processing eliminates writing to the .ruj file and to disk; enabling journal optimization eliminates most of the overhead in writing to the root file. The combination of fast commit and journal optimization can dramatically increase transaction processing speed.

Enabling journal optimization requires the following prerequisites:

- Fast commit processing must be enabled so that the database recovery process can use checkpoint information if recovery is necessary.
- After-image journaling must be enabled so that commit information can be written to the .aij file instead of the root file. After-image journaling is required by fast commit processing.
- Snapshots must be disabled or enabled deferred.
  - If snapshots are disabled and journal optimization is enabled, no information is written to the root file.
  - If snapshots are deferred and journal optimization is enabled, an active read-only transaction causes the journal optimization option to be *disabled* until the read-only transaction finishes.

The journal optimization option is intended for use on database systems that are update-intensive. Oracle Corporation does not recommend enabling this feature on systems that experience frequent read-only transactions for the following reasons:

- If snapshots are disabled, read-only transactions may generate lock contention, thereby slowing down transaction processing.
- If snapshots are deferred, read-only transactions may cause switching between journal optimization (not writing to the database root file) and the default (writing to the database root file). This switching results in expensive overhead. Furthermore, read-only transactions stall until all active read/write transactions commit.

Journal optimization conforms to the basic Oracle Rdb architecture, including the assignment and use of transaction sequence numbers (TSNs) to ensure database integrity and consistency. When the journal optimization option is disabled, TSNs are assigned to users one at a time; when the journal optimization option is enabled, each user is preassigned a range of TSNs.

Assigning a range of TSNs for each user avoids the single-threading problem because commit information need not be written to the database root for each transaction. All transaction information is written to the .aij file with the exception of each user's allocated TSN range. The allocated range is written to the root file. If a user runs out of TSNs, a new TSN range is assigned. The size of the TSN range is specified with the TRANSACTION INTERVAL IS *n* qualifier when you enable the journal optimization option, where *n* equals the number of TSNs. See Section 4.1.5.4 for syntax details.

The transaction interval value (the TSN range) must be a number between 8 and 1024. The default value is 256. You should consider the following points as you determine a transaction interval value:

- A large transaction interval value requires less I/O to update the root file (necessary when a user allocates a new TSN range) because a user runs out of TSNs less frequently than with a small interval value. Conversely, a low value requires more I/O to the root file. This I/O can degrade database performance.
- A large transaction interval value can cause processes to use their allocated TSNs inefficiently. Suppose the transaction interval is 512 and your database has 512 active users each day. This means that your database will use 262,144 TSNs each day, even though only a few of the 512 TSNs may actually be used. Furthermore, if the 512 users open and close the database four times each day, the database will use 1,048,576 TSNs each day. Thus a database could run out of available TSNs, requiring the TSN values to be reset.

  Conversely, the smaller the transaction interval value, the longer the time between TSN reset.

You need to decide which constraint has precedence on your database: performance or running out of TSNs. As a general guideline, if your database has few users or if all user sessions are long, select a high transaction interval. If your database has many users or if user sessions are short, select a smaller transaction interval.

You can determine if a transaction interval is optimum by selecting a value and monitoring I/O to the root file. Then raise or lower the interval value, monitor the rate for the new value, compare the results with the old value, and adjust the current setting accordingly.

4.1.5.4 Enabling Fast Commit Transaction Processing

You can enable fast commit processing, and optionally journal optimization, with the SQL ALTER DATABASE statement. The following example shows the syntax:

```
SQL> ALTER DATABASE FILENAME test1
cont>    JOURNAL FAST COMMIT {ENABLED | DISABLED}
cont>            (CHECKPOINT INTERVAL IS n BLOCKS,
cont>             CHECKPOINT TIMED EVERY s SECONDS
cont>             [NO] COMMIT TO JOURNAL OPTIMIZATION,
cont>             TRANSACTION INTERVAL IS m);
```

The value specified for *n* equals the number of blocks the .aij file can increase by before checkpointing. The value specified for *s* equals the number of seconds elapsed before checkpointing. The value specified for *m* equals the number of TSNs you want to preassign to each user if you have enabled journal optimization.

Note

The TRANSACTION INTERVAL IS parameter qualifies the journal optimization option and is *not* a third way to specify a checkpoint interval. Also, do not confuse the TRANSACTION INTERVAL IS parameter with the value specified by the logical name RDM$BIND_CKPT_TRANS_INTERVAL or the configuration parameter RDB_BIND_CKPT_TRANS_INTERVAL. RDM$BIND_CKPT_TRANS_INTERVAL and RDB_BIND_CKPT_TRANS_INTERVAL specify transaction count as the checkpoint interval.

You can specify either or both checkpoint parameters. The following example enables fast commit processing and specifies checkpoint intervals of 256 blocks, 120 seconds, and 20 transactions. The example also enables journal optimization and specifies a range of 512 TSNs for each user:

```
$ DEFINE RDM$BIND_CKPT_TRANS_INTERVAL 20
$ SQL
SQL> ALTER DATABASE FILENAME test1
cont>    SNAPSHOT IS DISABLED
cont>    JOURNAL FILENAME DISK2:[USER]test2.aij
cont>    JOURNAL FAST COMMIT ENABLED
cont>          (CHECKPOINT INTERVAL IS 256 BLOCKS,
cont>           CHECKPOINT TIMED EVERY 120 SECONDS
cont>           COMMIT TO JOURNAL OPTIMIZATION,
cont>           TRANSACTION INTERVAL IS 512);
```

Notice that, to enable fast commit processing, after-image journaling must be enabled; to enable journal optimization, snapshots must be disabled (or deferred). If AIJ is not enabled, fast commit and journal optimization are not activated. If snapshots are not disabled or deferred, journal optimization is not activated.

The following example enables fast commit, but not journal optimization:

```
SQL> ALTER DATABASE FILENAME test1
cont>    JOURNAL FAST COMMIT ENABLED
cont>          (CHECKPOINT INTERVAL IS 256 BLOCKS,
cont>           CHECKPOINT TIMED EVERY 120 SECONDS
cont>           NO COMMIT TO JOURNAL OPTIMIZATION);
```

You can disable fast commit processing or journal optimization but specify checkpoint intervals or a TSN range that can be activated when you enable fast commit and journal optimization at some other time. Note that you must enable fast commit to activate the journal optimization option. Although journal optimization appears to be enabled in the following example, it is effectively disabled because fast commit is disabled.

```
SQL> ALTER DATABASE FILENAME test1
cont>    JOURNAL FAST COMMIT DISABLED
cont>          (CHECKPOINT INTERVAL IS 256 BLOCKS,
cont>           CHECKPOINT TIMED EVERY 120 SECONDS
cont>           COMMIT TO JOURNAL OPTIMIZATION,
cont>           TRANSACTION INTERVAL IS 512);
```

You can enable both features and the parameters set in the last example if you enter the following statement:

```
SQL> ALTER DATABASE FILENAME test1
cont>    JOURNAL FAST COMMIT IS ENABLED
cont>      (COMMIT TO JOURNAL OPTIMIZATION);
```

Refer to the *Oracle Rdb7 SQL Reference Manual* for a complete description of fast commit processing syntax.

You can determine if fast commit processing is enabled by examining the Fast Commit Information screen in the Performance Monitor. The following example shows a Fast Commit Information screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 24-JUN-1996 11:43:21
Rate: 3.00 Seconds         Fast Commit Information        Elapsed: 00:00:32.56
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------

AIJ Fast Commit is enabled
- Checkpointing AIJ interval is 256 blocks
- Checkpointing time interval is 120 seconds
Commit to AIJ optimization is enabled
- Transaction interval is 512









--------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
```

The Fast Commit Information screen is located in the Database Parameter Information submenu.

You can also use the SHOW TRANSACTION statement to determine if fast commit processing is enabled. If you enable fast commit processing and use the SET TRANSACTION statement to start a read/write transaction reserving the EMPLOYEES table for shared read, the SHOW TRANSACTION statement displays the following information:

```
SQL> SHOW TRANSACTION

Transaction information:
    Statement constraint evaluation is off

On the default alias
Transaction characteristics:
        Read Write
        Reserving table EMPLOYEES for shared read
Transaction information returned by base system:
a read-write transaction is in progress
  - updates have not been performed
  - fast commit is enabled
  - aij commit is enabled
  - transaction sequence number (TSN) is 137
  - snapshot space for TSNs less than 137 can be reclaimed
  - session ID number is 82
SQL>
```

You can examine checkpoint statistics by using the Performance Monitor Checkpoint Statistics screen (refer to Section 4.1.1.11).

4.1.5.5 Memory Page Transfers

When the memory page transfers feature is enabled, a process does not need to write a modified page to disk before another process accesses the page. In other words, pages in a process' allocate set can contain committed updates from another process that have *not* been written to disk. If a database has the required characteristics for using memory page transfers, the performance gain of update-intensive applications can be significant. This is because of the number of inputs/outputs (I/Os) saved by not writing updates to disk and the ability to share pages among processes.

The memory page transfers feature is available for any database with the following characteristics:

- Global buffers are enabled.
- After-image journaling is enabled.
- Fast commit processing is enabled.
- The database can only be accessed from a single node (the NUMBER OF CLUSTER NODES value for the database is 1).

If a database has these characteristics, you can specify the PAGE TRANSFER VIA MEMORY clause of the SQL CREATE DATABASE, ALTER DATABASE, or IMPORT statement to enable memory page transfers for the database. If you specify the PAGE TRANSFER VIA MEMORY clause for a database that does not have the appropriate characteristics, the memory page transfers feature is not enabled.

Until Version 6.1 of Oracle Rdb, page transfers between processes were always performed via disk. The default for the PAGE TRANSFER clause is PAGE TRANSFER VIA DISK, and applications that performed acceptably in the past using this option can continue to do so.

In a database with fast commit processing enabled, Oracle Rdb writes checkpoint records to the after-image journal (.aij) file. Then if a transaction fails, Oracle Rdb rolls back the current, failed transaction and performs a redo operation of all the committed transactions since the last checkpoint record in the .aij file. When the memory page transfers feature is enabled for a database, the recovery process is different because Oracle Rdb does not write checkpoint records to the .aij file. Instead, internal data structures track the checkpoint virtual block number (checkpoint VBN) for each process. If the database needs to be recovered, Oracle Rdb starts the redo operation at the lowest checkpoint VBN in the .aij file.

See the ALTER DATABASE, CREATE DATABASE, and IMPORT statements in the *Oracle Rdb7 SQL Reference Manual* for more information on the PAGE TRANSFER clause.

4.1.6 Row (or Record) Fragmentation

A row becomes fragmented when a modification extends a row's physical length and insufficient free space exists on the page to hold the new expanded row. Fragmentation is particularly apparent when a text field in a row changes frequently. For example, consider the instance when a row contains many empty columns and the row is compressed, and the page size was determined based on the compressed row length. When the page is filled with these compressed rows and values are entered for these empty columns, these rows can no longer fit on the page and Oracle Rdb must fragment the row to the next available page in the logical area for the table. Pointers attached to the row's fragments indicate the location of other fragments. This structure allows Oracle Rdb to fill database pages more efficiently. However, retrieving fragmented rows expends more I/O time than retrieving whole rows because Oracle Rdb must locate all the row fragments and assemble them into a complete row.

For this reason, you should try to avoid row fragmentation. For example, look at the biggest row in each storage area of your database in its nominal, uncompressed form. If the row in one storage area is 1800 bytes, make your page size for this storage area large enough to accommodate the data and overhead; in this case, a 4-block page for this storage area should suffice.

You can use the RMU Analyze command to determine whether or not rows in your database are fragmented. Row fragmentation can occur when rows are changed in either of the following two ways:

- Storing a row

  Newly stored rows may be too large to fit on a single database page. If free space in an unused database page is not large enough to store the entire row, the row will be fragmented. If you specified a value that is too small for the storage area PAGE SIZE based on a previous row size, you will need to create a new storage area with the SQL CREATE STORAGE AREA statement, use the PAGE SIZE parameter to specify a new page size, and use the SQL ALTER STORAGE MAP statement to map all rows to the new storage area. This alternative is preferable to using the SQL EXPORT and IMPORT statements in which all storage area rows would be unloaded and reloaded.

When you create a new row, but supply data values for only a few columns in that row, that row requires less space on the database page than if you supply values for all columns. Later, when you retrieve that row and fill those empty columns with actual values, the row requires more space on the database page. Therefore, you should try to assign data values for the majority of the columns in the row.

- Modifying a row

  You can change data types and column sizes for any column in the database and because you can add columns to a row, you risk creating a new version of a row that is larger than all previous versions. When Oracle Rdb writes this new, larger version of the row back into the database page, it may split the row between two separate database pages if there is insufficient free space.

When you delete a row from the database, Oracle Rdb changes the line index entry on that page for that row. It marks the line index entry as empty, but reserves it for the transaction that deleted the row. When the transaction is terminated with a COMMIT statement, the empty space becomes permanent. It is then available to other transactions as long as the dbkey scope is not specified and therefore defaults to the SQL DBKEY SCOPE IS TRANSACTION option. If the transaction is rolled back, Oracle Rdb uses that empty space to insert the row again.

4.1.7 Specifying the Number of Recovery Buffers

The default number of recovery buffers is 20. Recovery-unit journaling occurs for each active user of the database application and cannot be controlled by the user. Therefore, there are as many .ruj files as active users of the database application. When a user's transaction is rolled back, that user's .ruj file is used to complete the rollback process. Similarly, but on a much broader scale, when a system failure occurs and database access ceases, Oracle Rdb uses all users' .ruj files to complete rollback (recovery) after the system comes back up for all update transactions not committed at the time of the failure.

The time that all of these database recovery processes (DBRs) take depends on the number of active users (number of .ruj files), the lengths of transactions (size of the .ruj files), and the number of recovery buffers (dedicated memory for recovery). If your update transactions are all relatively short, the recovery process will not take much time. If your update transactions are relatively long, the recovery process may take a while to complete. By setting the number of recovery buffers to as large a number as feasible, the entire recovery procedure will run much faster as more information will be loaded into the buffers with each I/O operation.

OpenVMS VAX OpenVMS Alpha

You should select a value large enough to fill the working set of the DBR process. Use the SQL ALTER DATABASE statement to adjust the buffer count to be nearly equal to the WSEXTENT of the DBR process. That is, set the WSQUOTA for the DBR process high enough so that a specified number of physical pages is guaranteed to be available for the DBR process if database recovery is necessary. Set the WSEXTENT for the DBR process as high as possible to ensure that the maximum number of physical pages specified will be available when needed. See Chapter 6 for information on database recovery in a VMScluster environment. See the OpenVMS documentation set for a more detailed discussion on guidelines for setting working set parameters and for tuning automatic working set adjustment parameters. ♦

See the *Oracle Rdb7 Guide to Database Maintenance* for information on changing the NUMBER OF RECOVERY BUFFERS option.

4.1.8 Allocation for the After-Image Journal File

The .aij files are disabled by default when you initially define a database with the SQL CREATE DATABASE statement. You must use the SQL ALTER DATABASE statement and specify a file name with the JOURNAL IS option. The default allocation space for .aij files is 0 blocks. There are three important considerations concerning .aij files:

- Space allocation
- Extending the space allocation
- Backing up an .aij file to keep from filling the AIJ disk

When you create the .aij file, you can specify an allocation and an extent. The allocation defines the physical file size that is allocated on disk.

The extent parameter not only defines the physical extension that will occur if the file needs to be physically extended, but also is used to define a logical end-of-file (EOF) within the physical file allocation. For example, assume an .aij file is created with allocation equal to 10,000 blocks and extent equal to 1000 blocks. The file will be managed in increments of 1000 blocks. That is, the file will be logically extended and initialized 10 times before it is actually physically extended. These logical extensions will show up in the Performance Monitor even though the file may not have been physically extended. Hence, the file is initialized only up to the logical EOF.

The key to prevent the AIJ disk from filling up is to specify that the RMU Backup After_Journal command be executed before the file gets physically extended. That is, if there is much AIJ activity, set the Threshold qualifier value to a value large enough to ensure that the file does not extend, but not so large as to fill the disk. For example, if you set the value using the RMU

Backup After_Journal command with the Threshold=7000 qualifier, the AIJ backup threshold is 7000 blocks. That causes the backup process to start backing up the .aij file in a catch up mode without blocking other transactions when the end of the .aij file is less than the Threshold value. Only if the end of the .aij file exceeds the Threshold value is a quiet-point lock requested. When the quiet-point lock is granted, transactions are blocked until the .aij file is completely backed up. The .aij file is backed up and truncated and the logical EOF is reset to zero. Because the logical EOF never reaches the physical EOF, the .aij file never gets physically extended and the disk never fills up. For more information on backing up the .aij file, see the *Oracle Rdb7 Guide to Database Maintenance*.

The backup process starts backing up the .aij file before the Threshold value is reached and does so without blocking other transactions. Only when the Threshold value is exceeded is a quiet-point lock immediately requested and when granted, no active transactions are allowed. Therefore, it is possible that a transaction could cause the .aij file to fill and furthermore, physically extend this file before the transaction is done. This is where the extent parameter comes in. If the extent parameter is too large, the physical allocation of the file becomes too large. If the extent parameter is too small, the excessive extents cause some performance degradation.

For applications with high transaction volumes, setting a large allocation size to minimize extents can improve performance significantly. For applications with a large number of updates and severe response time constraints, preallocate the .aij file to prevent it from being extended during normal processing. You can set the size large enough to handle processing for one day and use the RMU Backup After_Journal command once a day to unload the .aij file to tape. If you have one disk device dedicated to the .aij file, set the allocation size to take up the entire disk.

4.1.9 Allocation for Snapshot Files

You can specify a database-wide size allocation for your snapshot files with the SQL CREATE DATABASE statement or specify it individually for each storage area with the SQL CREATE STORAGE AREA statement.

First determine if the default snapshot allocation of 100 pages to be applied database-wide is large enough to handle the needs of your read-only users for all storage areas without extending. Or you can determine the frequency with which each storage area is being accessed by read-only users as opposed to read/write users and determine if the default allocation of 100 pages should be adjusted accordingly. Remember that update transactions write before-images of each row they update to a snapshot file. So, if you determine the transaction rate and update activity for the tables in a storage area, the results give you an estimate of the relative growth and potential size of the snapshot file. Set the allocation accordingly for these storage areas.

If the snapshot file grows too large, you can truncate it to a smaller size by specifying a smaller allocation value using the SNAPSHOT ALLOCATION IS option of the SQL ALTER DATABASE statement for the RDB$SYSTEM storage area snapshot file. You can alternatively use the SNAPSHOT ALLOCATION IS option of the SQL ALTER DATABASE ALTER STORAGE AREA statement to modify snapshot allocation sizes for other storage areas. Another way to control the growth of snapshot files is to set them to deferred; use the SNAPSHOT IS ENABLED DEFERRED option of the ALTER DATABASE statement. In this way, read/write transactions only write before-images to the snapshot file when a read-only transaction is in progress.

4.1.10 Extents for After-Image Journal and Snapshot Files

For performance reasons, you should avoid extents by setting the allocation for .aij and .snp files sufficiently large at the beginning. However, if extents occur, you should select extent options to avoid multiple extents that further degrade performance. You can accomplish this by specifying an extent value of sufficient size so that one extent handles the maximum expansion of the file.

4.1.11 Accessing the Snapshot File

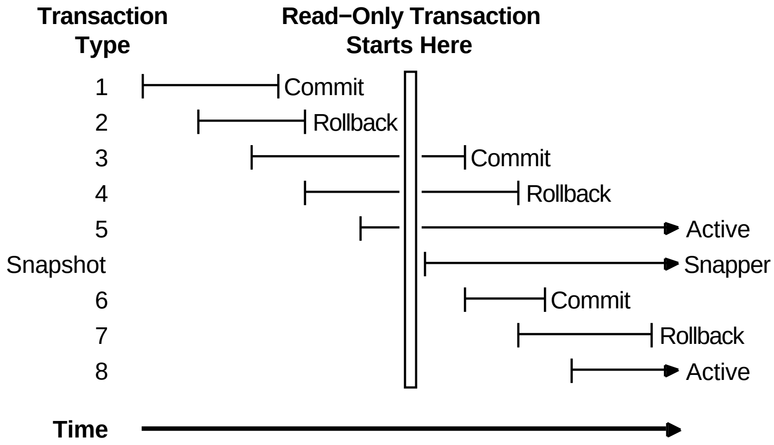
By including READ ONLY in your SQL SET TRANSACTION statement, you have access to the snapshot file of the database. A read-only transaction provides a consistent view of the database (a snapshot in time), as of the start of the transaction. Because only shared or protected writers (not exclusive writers) place before-images of anything they modify in the .snp file, a read-only transaction can see those images without waiting for the writers to release locks. Also, read-only transactions need not hold read locks on data to maintain a consistent view. So the benefits of read-only transactions are:

- Consistent view of data at a single point in time
- Minimal lock conflict

To provide a read-only transaction with the correct version of the row (the version that has been committed before the read-only transaction started), Oracle Rdb uses TSNs. Oracle Rdb assigns a TSN to every transaction that starts. If the transaction updates the database, the TSN of that transaction is attached to the new version of the row.

Figure 4–10 shows a read-only transaction time line and how the read-only transaction views various types of transactions.

Figure 4–10 Transactions Accessing the Database Snapshot File



ZK–7537–GE

The snapshot file shows updates only for transactions that had already been committed when the read-only transaction started. These include only type 1 transactions.

The transactions whose updates do not appear in the snapshot file can be placed into one or more of the following categories:

- Transactions that were still active at the time the read-only transaction started. These include types 3, 4, and 5.
- Transactions that started after the read-only transaction started. These include types 6, 7, and 8.
- Transactions that were rolled back. This includes type 2.

4.1.12 Making the Snapshot File Optional for Oracle Rdb

When large database update programs or heavy multiuser update activity occurs, all I/O operations necessary to complete the transactions may affect general database performance. Oracle Rdb allows load or update programs that perform many changes to rows in the database to take full advantage of the I/O operation capabilities of a particular disk device. By disabling write operations to the snapshot file, all I/O operations are concentrated on access to the database itself.

Note

You should never delete a snapshot (.snp) file because it will corrupt the database following a database backup and restore operation. If you delete an .snp file, the pointers to the .snp file in the database root (.rdb) file are not updated. Always disable snapshot files with an SQL ALTER DATABASE statement if you do not want to use them.

When concurrent update access to the database is particularly heavy, Oracle Rdb must write to the database, the .snp file, the .ruj file, and the .aij file, if after-image journaling is enabled. You can eliminate writes to the snapshot file by disabling snapshots, and reduce the I/O operations necessary to complete each transaction. You can temporarily disable snapshots by using the exclusive option when updating. Disabling snapshots should be a last resort for improving performance for applications that are update-oriented. As another option, you can place the .snp files on different disks from their storage area (.rda) files. In almost all situations, the advantages of avoiding locking and deadlock situations through the use of snapshot files far outweighs the I/O overhead of snapshot files.

The following statements allow you to enable or disable user access and database write operations to the .snp file when you specify the SNAPSHOT IS DISABLED option:

- SQL CREATE DATABASE
- SQL ALTER DATABASE
- SQL IMPORT

When snapshot files are enabled, a read/write transaction that modifies data records writes a before-image of the modified records to the snapshot files. If a read-only transaction starts after the read/write transaction and tries to read any of the records modified by the read/write transaction, Oracle Rdb instead shows the read-only transaction the before-images of the modified records from the snapshot file. The read-only transaction does not use any locks to read any records.

When snapshot files are disabled, the behavior of both read-only and read/write transactions is different than when snapshot files are enabled. When snapshot files are disabled, read/write transactions do not write before-images of modified records to the snapshot files; thus there are no before-images of modified records for read-only transactions to read. Instead, a read-only transaction must read actual data records as a read/write transaction does, which requires that the read-only transaction use locks (as a read/write transaction does) to get a consistent view of the data records. A read-only transaction in a database with snapshot files disabled can be thought of as a special type of read/write transaction; it locks data records like a read/write transaction, but it still cannot update the database. Read-only transactions in databases with snapshot files disabled get shared read locks to data records, which enables all other users to read any records being read by the read-only transactions.

When snapshots are disabled, actual read/write transactions can modify data records and until the modifications are committed or rolled back, any other transaction trying to access these data records will encounter a lock conflict. Also, you may be prevented from accessing certain areas of the database because an update transaction has locked index nodes that you require to retrieve a set of rows from another table.

When other update transactions update rows in a table sequentially, Oracle Rdb may place locks on the entire table for that transaction until that transaction terminates. Therefore, retrieval operations that specify read-only when snapshots are disabled may not execute in the same way as when snapshots are enabled.

Remember, you disable snapshots to increase performance for certain classes of database update operations where multiuser access by large numbers of users retrieving rows is not critical. Such operations may be batch load, transaction processing, or update programs that are scheduled to execute when interactive access is not heavy. When these update operations complete and efficient concurrent access is required again, you can enable snapshots. If no multiuser access is required during the database update operation, then it is probably better to use an exclusive transaction for the operation. Using an exclusive transaction provides optimal performance for the update operation, and you do not need to disable snapshot files using the SQL ALTER DATABASE or IMPORT statement.

The SQL CREATE DATABASE statement in Example 4–14 creates a database file, DR2:[RDB]mf_personnel.rdb, and, by default, creates and enables the .snp file, DR2:[RDB]mf_pers_default.snp, for the defined storage area RDB$SYSTEM.

Example 4–14 Creating a Snapshot File for the RDB$SYSTEM Storage Area

```
SQL> CREATE DATABASE FILENAME 'DR2:[RDB]mf_personnel'
cont> CREATE STORAGE AREA RDB$SYSTEM FILENAME 'DR2:[RDB]mf_pers_default.rda';
```

The SQL CREATE DATABASE statement in Example 4–15 includes an explicit qualifier that creates and enables the .snp file.

Example 4–15 Creating and Explicitly Enabling a Snapshot File for the RDB$SYSTEM Storage Area

```
SQL> CREATE DATABASE FILENAME 'DR2:[RDB]mf_personnel' SNAPSHOT IS ENABLED;
```

If you do not have sufficient space to contain all the possible database files or if you do not need the .snp file, you may choose a small initial allocation size for the .snp file, or you can set the snapshot allocation to 0 pages. Two cases where setting the snapshot allocation to 0 is useful are:

- If you have disabled snapshots and you want to save some space
- If you have changed a read/write storage area to read-only, and you want to save some additional space because the snapshot file is not used

Using the first option in the SQL CREATE DATABASE statement, you can control the amount of disk space used by the .snp file as shown in Example 4–16.

Example 4–16 Creating a Snapshot File for the RDB$SYSTEM Storage Area—Limiting Size to 50 Pages

```
SQL> CREATE DATABASE FILENAME 'DR2:[RDB]mf_personnel'
cont>   SNAPSHOT IS ENABLED
cont>   CREATE STORAGE AREA RDB$SYSTEM FILENAME 'DR2:[RDB]mf_pers_default.rda'
cont>      SNAPSHOT ALLOCATION IS 50 PAGES
cont>      SNAPSHOT EXTENT IS 200 PAGES;
```

These SQL CREATE DATABASE and CREATE STORAGE AREA statements do the following:

- Create a database file DR2:[RDB]mf_personnel.rdb
- Enable writing to the .snp file
- Create an .rda file DR2:[RDB]mf_pers_default.rda
- Create an .snp file for the mf_pers_default.rda storage area DR2:[RDB]mf_pers_default.snp
- Set the allocation of the .snp file to 50 blocks
- Set the extent size for the .snp file to 200 pages

Example 4–17 uses the SQL ALTER DATABASE statement to modify two database-wide parameters and do the following:

- Disable writing to .snp files
- Enable after-image journaling by specifying an .aij file name

The SQL ALTER DATABASE statement modifies the allocation and extent size of the .snp file as shown in Example 4–17.

Example 4–17 Modifying Snapshot File Allocation and Extent Size

```
SQL> ALTER DATABASE FILENAME mf_personnel.rdb
cont>    SNAPSHOT IS DISABLED
cont>    JOURNAL FILE DR3:[RDB]newpers.aij
cont>    SNAPSHOT ALLOCATION IS 3 PAGES
cont>    SNAPSHOT EXTENT IS 200 PAGES;
```

This SQL ALTER DATABASE statement does the following:

- Disables snapshots
- Creates and enables a new .aij file, DR3:[RDB]newpers.aij
- Specifies a smaller .snp file allocation to truncate the .snp file

  If the current .snp file allocation for the RDB$SYSTEM storage area is larger than the size you requested, Oracle Rdb truncates the file. Therefore, you can reclaim disk space by allocating a smaller allocation size for the .snp file.
- Specifies the snapshot file extent size

Example 4–18 shows you how to specify snapshot extents for multivolume databases using the SQL ALTER DATABASE statement.

Example 4–18 Specifying Snapshot Extents for a Multivolume Database

```
SQL> ALTER DATABASE FILENAME mf_personnel.rdb
cont>    SNAPSHOT IS ENABLED
cont>    JOURNAL FILE DR3:[RDB]newpers.aij
cont>    SNAPSHOT ALLOCATION IS 200 PAGES
cont>    SNAPSHOT EXTENT IS (MINIMUM OF 200 PAGES,
cont>    MAXIMUM OF 1000 PAGES, PERCENT GROWTH IS 15);
```

4.1.13 Deferred Snapshots Capability

In general, enabling snapshots improves performance when a database includes concurrent read-only and read/write transactions. The read-only user can retrieve data without locking rows that other users want to access, and can see a consistent version of the data. That is, the read-only user can read rows that were updated by any transaction that committed before the read-only transaction started but does not see any changes made to rows while the read-only transaction is active. Performance improves because there is less contention in a concurrent environment where read-only users would normally encounter or generate record lock conflicts.

However, when snapshots are enabled, a shared or protected read/write transaction must write a copy of the row to be updated to the .snp file. The extra I/O operation overhead incurred by each read/write transaction is worthwhile to a database application if read-only users are accessing the database. If no read-only users are accessing the database (or if there are very few read-only users and their transactions are short), you can reduce the number of I/O operations to the snapshot file by enabling deferred snapshots. When deferred snapshots are enabled, writing to the snapshot file is deferred until a read-only transaction starts.

Note

The performance benefit of read-only transactions is that they do not generally lock out, or get locked out by, read/write transactions (except in SQL READ WRITE . . . RESERVING <table-name> EXCLUSIVE transactions). In other words, if your performance problem is locking, then enabling snapshots helps. If the bottleneck is disk I/O operations, then enabled snapshots may hurt performance. By moving the .snp file to a different disk, you may remove the disk I/O operation bottleneck. You could also try enabling global buffers to reduce the disk I/O. If these actions do not improve performance, then deferred snapshots *may* be the answer. If disk I/O operation remains the problem, but locking is not a problem, disabling snapshots might provide a solution. See Section 8.1.3.6 for more information on snapshot files and performance.

The deferred snapshot feature is supported by the SNAPSHOT ENABLED DEFERRED option in the SQL CREATE DATABASE, ALTER DATABASE, and IMPORT statements. See the *Oracle Rdb7 SQL Reference Manual* or online SQL help file for the syntax diagrams and reference information.

The default is SNAPSHOT ENABLED IMMEDIATE. When snapshots are immediate, a read/write transaction always writes a before-image of the row it is about to update to the .snp file. A read-only transaction (with the reserving option or without it) always attempts to read a row in the data file first. But if the row the read-only user wants to access is marked for update, the read-only user reads the before-image of the row in the .snp file. When the read/write user's transaction ends, two things can happen:

- If the read/write user issues a COMMIT statement, the updated row is written to the data file; however, active read-only users cannot see the change because they continue to read the version of the row in the .snp file.
- If the read/write user enters a ROLLBACK statement, the before-image copy of the row is retrieved from the read/write user's .ruj file. Active read-only transactions read the version of the row in the data file because no changes have been applied.

If snapshots are deferred, then a read/write transaction needs to write to the snapshot file only if read-only transactions are attached to the database when the read/write transaction begins. If someone tries to start a read-only transaction while read/write transactions are active, the read-only transaction is forced to wait for the read/write to commit or roll back before it can begin. The read-only transaction is forced to wait because the active read/write transaction is not writing to the .snp file. If another read/write transaction starts while the read-only transaction is waiting, this second read/write transaction writes before-images to the .snp file.

Table 4–7 represents one possible SNAPSHOTS DEFERRED scenario.

Table 4–7 Transaction Behavior with Deferred Snapshots

Transaction	Write to Snapshot?	Comments
No users		
Transaction A starts read/write	No	
Transaction B starts read/write	No	
Transaction C starts read-only		Must wait
Transaction D starts read/write	Yes	
Transaction A commits		
Transaction B commits		

(continued on next page)

Table 4–7 (Cont.) Transaction Behavior with Deferred Snapshots

Transaction	Write to Snapshot?	Comments
Transaction C proceeds		
Transaction E starts read/write	Yes	
Transaction C commits		
Transaction F starts read/write	No	

The three snapshot modes can be summarized as follows:

- SNAPSHOT DISABLED

  Transactions do not write to an .snp file. Read-only transactions are converted to read/write. This results in lower I/O overhead for heavy update environments but can lead to lower concurrency and more lock contention because converted read-only transactions generate locks just like read/write transactions.

- SNAPSHOT ENABLED IMMEDIATE

  Shared and protected read/write transactions write before-images to the snapshot file regardless of whether or not read-only transactions are accessing the database. Read-only transactions never encounter locks because old versions of updated rows are kept in the .snp file. This results in higher concurrency but also more I/O.

  Note that exclusive and batch-update transactions never write to an .snp file because, by definition, they do not allow concurrency. See Section 3.8.3 and Section 3.8.3.5 for more information.

- SNAPSHOT ENABLED DEFERRED

  When only read/write users are active, SNAPSHOT ENABLED DEFERRED behaves like SNAPSHOT DISABLED; nothing is written to the .snp file. Read-only transactions stall until all previously started read/write transactions complete (they wait because the read/write transactions did not write to the snapshot file). Once a read-only transaction starts, the database behaves like the SNAPSHOT ENABLED IMMEDIATE mode. When all read-only transactions have completed, active read/write transactions continue writing to the .snp file but new read/write transactions do not.

The SNAPSHOT ENABLED DEFERRED mode provides a balance between having snapshots disabled and enabled immediate. This mode especially benefits environments that fluctuate between update-intensive and read-intensive. For example, a database may receive many updates during the day (but few reads). Then during off hours, batch applications may read the updates and generate reports.

To determine the current snapshot setting for a database, use the Performance Monitor General Information screen. Example 4–19 shows an example of a General Information screen. You can also use the SQL SHOW DATABASE statement to display snapshot information.

Example 4–19 Determining the Current Setting for Snapshot Files

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 24-JUN-1996 13:20:16
Rate: 3.00 Seconds            General Information           Elapsed: 00:00:13.03
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------

Database created at 30-MAY-1996 14:20:42.77
Maximum user count is 50
Maximum node count is 16
Database open mode is Automatic
Database close mode is Automatic
Snapshot mode is Deferred
Statistics collection is enabled
Active storage area count is 10
Reserved storage area count is 0
Default recovery-unit journal filename is "Not Specified"
Date of last backup is 17-NOV-1858 00:00:00.00
Fast incremental backup is enabled




--------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
```

The General Information screen is located in the Database Parameter Information submenu.

4.2 Adjusting Storage Area Parameters

The following sections describe the storage area parameters and provide information you can use to determine optimum values for your database applications. Because you can often attain good performance with Oracle Rdb using the default values for each storage area parameter, you should use these sections as guidelines to adjust your database for applications with unusual characteristics. For example, if your application relies heavily on the segmented string data type, you may be able to improve performance by placing the segmented strings in their own storage area with mixed page format, specific SPAM thresholds and intervals, page size, and so forth.

Furthermore, if analysis of the database shows row fragmentation, you may be able to reduce fragmentation by specifying certain database parameter values using the SQL ALTER DATABASE, EXPORT, and IMPORT statements. See Section 4.1.6 on row or record fragmentation for a complete discussion of identifying and remedying a database with many fragmented records.

Table 4–8 summarizes the Oracle Rdb default values for storage area parameters.

Table 4–8 Oracle Rdb Multifile Storage Area Parameter Values: Default, Minimum, and Maximum

Storage Area Parameters	Default Values (Minimum/Maximum)
Storage area name	RDB$SYSTEM
Storage area file name	Must be specified (.rda)
Allocation	400 pages (1/disk device size)
Page size	2 blocks (1 block/32 blocks)
Page format	Uniform
SPAM thresholds (mixed format)	70%, 85%, 95%
SPAM thresholds (uniform format)	0%, 0%, 0%
SPAM interval	216 pages[1]
Storage area extent pages	100 pages (0/disk device size)
Storage area extent options	Minimum 99 pages, maximum 9,999 pages, growth 20%
Snapshot file name	Same as storage area file (.snp)
Snapshot file allocation	100 pages (0/disk device size)
Snapshot file extent pages	100 pages (0/disk device size)
Snapshot file extent options	Minimum 99 pages, maximum 9,999 pages, growth 20%

[1] (Minimum: 216/Maximum: ((Blocks-per-page * 512)–22) * 4)

Use the SQL ALTER DATABASE statement and CREATE STORAGE AREA clause to define new storage areas with new storage area and file names when database changes are needed. The changes best implemented in this way are: changing page size, SPAM intervals and thresholds, snapshot file name, and page format. Once the new storage area is defined, modify the STORE clause in the SQL ALTER STORAGE MAP statement to point to the new storage area. Data is unloaded from the old storage area and loaded into the new storage area. To complete the operation, delete the old storage areas with the SQL DROP STORAGE AREA clause within the ALTER DATABASE statement.

Table 4–9 describes specific SQL statements and if each statement allows you to specify storage area parameters, using the SQL CREATE DATABASE, ALTER DATABASE, or IMPORT statement, respectively.

Table 4–9 SQL Statements Affecting Storage Area Parameters

Storage Area Parameters	SQL CREATE STORAGE AREA	SQL ALTER STORAGE AREA	SQL IMPORT or CREATE STORAGE AREA
Specify storage area name	Yes	Yes	Yes
Specify storage area RDB$SYSTEM	Yes	No	Yes
Specify storage area file name	Yes	No	Yes
Allocation	Yes	No	Yes
Page size	Yes	No	Yes
Page format	Yes	No	Yes
Specify SPAM thresholds	Yes	No	Yes
Specify SPAM interval	Yes	No	Yes
Storage area extent pages	Yes	Yes	Yes
Storage area extent options	Yes	Yes	Yes
Snapshot file name	Yes	No	Yes
Snapshot file allocation	Yes	Yes	Yes
Snapshot file extent pages	Yes	Yes	Yes
Snapshot file extent options	Yes	Yes	Yes
Specify the read attribute	No	Yes	No

Table 4–10 describes the specific Oracle RMU commands that allow you to specify storage area parameters.

Table 4–10 Oracle RMU Commands Affecting Storage Area Parameters

Storage Area Parameters	RMU Restore	RMU Copy_Database	RMU Move_Area
Specify storage area name	No	No	No
Specify storage area RDB$SYSTEM	No	No	No
Specify storage area file name	Yes	Yes	Yes
Allocation	No	No	No
Page size	Yes	Yes	Yes
Page format	No	No	No
Specify SPAM thresholds	Yes	Yes	Yes
Specify SPAM interval	No	No	No
Storage area extent pages	Yes	Yes	Yes
Storage area extent options	No	No	No
Specify snapshot file name	Yes	Yes	Yes
Snapshot file allocation	Yes	Yes	Yes
Snapshot file extent pages	No	No	No
Snapshot file extent options	No	No	No
Specify the read attribute	No	No	No

See the *Oracle Rdb7 Guide to Database Maintenance* and the *Oracle RMU Reference Manual* for more information on using the RMU Restore, the RMU Copy_Database, and the RMU Move_Area commands to modify specific storage area parameters.

4.2.1 Gathering Storage Area Parameter Information

Sections 4.2.1.1 through 4.2.1.4 describe RMU Analyze commands and Performance Monitor screens that you can use to gather storage area information. For general information on using the RMU Analyze command, refer to Section 2.1. For general information on using the Performance Monitor, refer to Section 2.2.

4.2.1.1 RMU Analyze Areas Command

This section describes the format and content of the output when you use the RMU Analyze command and specify the Areas [= storage-area-list] and the Option [= Normal, Full, or Debug] qualifiers.

Using the RMU Analyze Areas Option=Normal Command

When you use the RMU Analyze Areas command and specify the Option=Normal qualifier to gather information on the mf_personnel database, Oracle Rdb displays a summary of important information for the storage area and each of the logical areas defined for the storage area, as shown in Example 4–20.

Example 4–20 RMU Analyze Areas Command with the Option=Normal Qualifier

```
$ RMU/ANALYZE/AREAS=EMPIDS_LOW/OPTION=NORMAL mf_personnel
----------------------------------------------------------------------------
 Storage analysis for storage area: EMPIDS_LOW - file: $DUA0:[ORION]EMPIDS_LOW.RDA;1
 Area_id: 2,  Page length: 1024,  Last page: 52

 Bytes free: 35992 (68%), bytes overhead: 6111 (11%)
 Spam count: 1, AIP count: 0, ABM count: 0
 Data records: 222, bytes used: 11145 (21%)
   average length: 50, compression ratio:  .87
   index records: 83, bytes used: 4768 (9%)
     B-Tree: 0, Hash: 1978, Duplicate: 2790, Overflow: 0
-----------------------------------
 Logical area: RDB$SYSTEM_RECORD for storage area : EMPIDS_LOW
 Larea id: 57,  Record type: 0, Record length: 215, Not Compressed

 Data records: 0, bytes used: 593 (1%)
-----------------------------------
 Logical area: EMPLOYEES_HASH for storage area : EMPIDS_LOW
 Larea id: 58,  Record type: 0, Record length: 215, Not Compressed

 Data records: 26, bytes used: 989 (2%)
   average length: 38
-----------------------------------
 Logical area: EMPLOYEES for storage area : EMPIDS_LOW
 Larea id: 63,  Record type: 26, Record length: 121, Compressed

 Data records: 37, bytes used: 2723 (5%)
   average length: 74, compression ratio:  .63
-----------------------------------
```

(continued on next page)

Example 4–20 (Cont.) RMU Analyze Areas Command with the Option=Normal Qualifier

```
 Logical area: JOB_HISTORY_HASH for storage area : EMPIDS_LOW
 Larea id: 66,  Record type: 0, Record length: 215, Not Compressed

 Data records: 26, bytes used: 989 (2%)
   average length: 38
------------------------------------
 Logical area: JOB_HISTORY for storage area : EMPIDS_LOW
 Larea id: 69,  Record type: 29, Record length: 42, Compressed

 Data records: 102, bytes used: 3654 (7%)
   average length: 36, compression ratio:  .97

-------------------------------------------------------------------------
```

The EMPIDS_LOW storage area shown in Example 4–20 has four logical areas as defined by the SQL CREATE INDEX and SQL CREATE STORAGE MAP statements' store clauses:

- EMPLOYEES_HASH hashed index
- EMPLOYEES table
- JOB_HISTORY_HASH hashed index
- JOB_HISTORY table

The summary information for the EMPIDS_LOW storage area is shown in the following list. Entries in parentheses that follow each field description are keyed to Example 4–20.

- Area_id

  The storage area ID assigned when the storage area was created (2).

- Page length

  The length of the page in bytes (1024).

- Last page

  The last page number in the storage area (52).

- Bytes free

  The total number of bytes available for storing data in the storage area and the percentage of free space (35,992 bytes, (68%) free).

- Bytes overhead

The total number of bytes of overhead used in the storage area and the percentage of space this value represents (6111 bytes, (11%) used).

- Spam count

  The total number of SPAM pages in the storage area (1).

- AIP count

  The total number of AIPs in the storage area (0).

- ABM count

  The total number of ABM pages in the storage area (0).

- Data records

  The total number of data rows in all logical areas (222). This number is the sum of the following:

  - The total of the “Data records” counts from all the logical areas in the storage area
  - The number of duplicate node records for all the hashed indexes from all the logical areas in the storage area
  - The number of index nodes below the index root (including duplicate nodes for non-ranked indexes and overflow nodes for ranked indexes) for all the sorted indexes from all the logical areas in the storage area

- Bytes used

  The total number of data bytes used in the storage area and the percentage of space this value represents (11,145 bytes, (21%) used).

- Average length

  The average row length for all records in the storage area (50).

- Compression ratio

  The average compression ratio (compressed/uncompressed) for all rows in the storage area (.87).

- Index records

  The total number of index records for all indexes in the storage area (83). For hashed indexes, this is the number of hash buckets and duplicate node records. For sorted indexes, this is the number of index nodes (including duplicate nodes for non-ranked indexes and overflow nodes for ranked indexes) below the root node.

- Bytes used

The total number of bytes used by all index records and the space this value represents for the storage area (4768 bytes, (9%) used).

- B-Tree, Hash, Duplicate, Overflow

  The total number of bytes used for each of these types of records (B-tree is 0, Hash is 1978, Duplicates is 2790, Overflow is 0).

The information for the EMPLOYEES_HASH logical area is shown in the following list. Entries in parentheses that follow each field description are keyed to Example 4–20.

- Logical area name

  The name of the logical area (EMPLOYEES_HASH).

- Storage analysis for storage area

  The storage area name (EMPIDS_LOW).

- Larea id

  The ID number assigned to the logical area when it is created (58).

- Record type

  The table ID; indexes are always 0 (0).

- Record length

  The maximum size of a record (uncompressed) plus overhead for this type of record (215). This value is meaningless for logical areas that describe indexes in mixed storage areas.

- Compression setting

  Record compression—indicates whether records are compressed or not compressed (Not compressed).

- Data records

  The total number of data records in the logical area (26). This total includes hash buckets for hashed indexes, root nodes for sorted indexes, and data rows for tables. For hashed indexes, this total does not include duplicate node records. For sorted indexes, this total does not include index nodes below the index root (including duplicate nodes and overflow nodes).

- Bytes used

  Total number of bytes of storage and percentage space used by all data rows (hash buckets for hashed indexes; root nodes for sorted indexes; data rows for tables) in the logical area. This value does not include any overhead; it is the same as the SIZE column on a per record basis in the output display when you specify the Option=Debug qualifier (989 bytes, (2%) used).

- Average length

  The average compressed or uncompressed length of all records (hash buckets, root nodes, and table data rows) in the logical area (38).

- Compression ratio

  The average compression ratio (compressed/uncompressed) for all records. This statistic is for tables only; because hashed index records are not compressed, no value is displayed.

In Example 4–20, the EMPIDS_LOW storage area, several statistics are important and it is worthwhile to keep track of them:

- The bytes free and percent free space in the storage area

  There are 35,992 bytes free; 68 percent of the storage area is available for storing more records. Using the average record length value (50 bytes), you can get a relative estimate of the number of records that can fit in the storage area (719 more records). This is a quick relative estimate. To be more precise, you can estimate the record number based on the average record length for each record in the logical area.

  When you are clustering rows, estimating available storage becomes more complicated because you are storing both parent and child rows together on the same page, making entries in the hash buckets, and perhaps creating more hash buckets. If you know the number of duplicates for each parent row and how many more parent rows you are storing, you can calculate approximately how many more parent and child rows could be stored in the table. You must also consider the hash structure and the amount of space it uses. The Option=Debug qualifier described later in this section gives you this information.

  The set of histograms displayed with the Option=Full qualifier (described later in this section) can help you to understand the way in which your database is growing.

  When you compare the percent bytes free to percent bytes used plus percent overhead bytes from different runs of the analysis output during the life of your database, you can determine how the database is growing and changing.

- The data record bytes used and overhead bytes used, and the percent space these values represent

The 222 total data rows in the storage area use 21 percent of the space in the storage area. By inspecting each logical area and counting the number of records, you can determine the following information:

- Thirty-seven EMPLOYEES rows use 2723 bytes (5 percent) and 102 JOB_HISTORY rows use 3654 bytes (7 percent), for a total of 12 percent of the space in the storage area.
- The 26 hash buckets in each hashed index logical area (EMPLOYEES_HASH and JOB_HISTORY_HASH) total 52 hash buckets that use 1978 bytes. There are 83 index records. The difference, (83 – 52 = 31), is the number of duplicate node records that use 2790 bytes. Both hash structures use 4768 bytes or 9 percent of the storage area.
- The 6111 bytes of overhead represent 11 percent of the space used in the storage area. Together, data rows (12 percent), index records (9 percent), and overhead (11 percent) use 32 percent of the space in the storage area.

From this information, an additional statistic, *proportional space use*, for logical areas and overhead can be calculated for a storage area. This statistic is a relative estimate of the proportional amount of space each logical area uses within the storage area along with the overhead for the area. This statistic can be used to project space use by logical areas in a storage area with the assumption that the nature and distribution of the records remain nearly the same. When this statistic is calculated on a regular basis with each analysis, it can indicate changes in the nature and distribution of space used by records for each logical area.

Proportional space use can be calculated for each logical area and overhead in the following way: compare the percent space each logical area uses in addition to the overhead; the ratio in this example is 5:2:7:2:11. This ratio represents the ratio of space used by 37 EMPLOYEES data rows (5 percent) to 26 EMPLOYEES_HASH hashed index records (2 percent) to 102 JOB_HISTORY data rows (7 percent) to 28 JOB_HISTORY_HASH hash index records (2 percent) to overhead (11 percent), respectively. From this comparison, the proportional space use ratios are as follows:

- For EMPLOYEES data rows to EMPLOYEES_HASH hash buckets, 2.5:1
- For JOB_HISTORY data rows to JOB_HISTORY_HASH hash buckets and duplicate node records, 3.5:1
- For EMPLOYEES and EMPLOYEES_HASH data rows to JOB_HISTORY and JOB_HISTORY_HASH data rows, 0.78:1

- For all logical area records to overhead, 1.45:1

These ratios can be useful in estimating future space utilization in the storage area if the nature and distribution of duplicate records do not change significantly. For overhead space use, these are relative estimates in that for small databases such as the mf_personnel database, the amount of overhead seems large. As the database grows, the overhead space use does not increase at the same rate as the database. You can regularly issue the RMU Analyze command with the Areas qualifier to calculate this rate of increase for overhead space use relative to the increasing space utilization by all logical areas within storage areas in the database.

Using the RMU Analyze Areas Option=Full Command

When you use the RMU Analyze Areas command and specify the Option=Full qualifier to gather information on the mf_personnel database, Oracle Rdb displays the same information in format and content that displays for the Option=Normal qualifier and, in addition, three types of histograms:

- Storage area page space use by page histogram

  This histogram summarizes the percentage of page space used by data for all logical areas compared with the total number of pages in the storage area. Page use is categorized into deciles. Values in parentheses are the total number of pages that fit within the decile category.

- Logical area page space use by page histogram

  This histogram summarizes the percentage of page space used ((used /used+free)*100) by data for a specific logical area compared with the number of pages that contain data. Page use is categorized into deciles. Values in parentheses are the total number of pages that fit within the decile category.

- Logical area percent of maximum record length by record histogram

  This logical area histogram summarizes each record's percentage of maximum size (uncompressed) for its record type. Percentage of maximum record length is categorized into deciles. Values in parentheses are the total number of records that fit within the decile category.

When you use the RMU Analyze Areas command with the Option=Full qualifier to analyze the EMPIDS_LOW storage area, Oracle RMU displays the following information:

- Summary information
- The storage area page space use by page histogram (labeled *% used vs # pages*) for the EMPIDS_LOW storage area

- Logical area information
- Pair of logical area histograms (labeled *used / used+free vs # pages* and *% of max length vs # of records*) for each of four logical areas, the EMPLOYEES_HASH hashed index, the EMPLOYEES table, the JOB_HISTORY_HASH hashed index, and the JOB_HISTORY table.

This information is shown in Example 4–21.

Example 4–21 RMU Analyze Areas Command with the Option=Full Qualifier

```
$ RMU/ANALYZE/AREAS=EMPIDS_LOW/OPTION=FULL mf_personnel
------------------------------------------------------------------------------
 Storage analysis for storage area: EMPIDS_LOW - file: $DUA0:[ORION]EMPIDS_LOW.RDA;1
 Area_id: 2,  Page length: 1024,  Last page: 52

 Bytes free: 35992 (68%), bytes overhead: 6111 (11%)
 Spam count: 1, AIP count: 0, ABM count: 0
 Data records: 222, bytes used: 11145 (21%)
   average length: 50, compression ratio:  .87
   index records: 83, bytes used: 4768 (9%)
     B-Tree: 0, Hash: 1978, Duplicate: 2790, Overflow: 0


                               % used vs # pages

  >90% | (0)
80-90% | (0)
70-80% |======== (4)
60-70% |== (1)
50-60% |====== (3)
40-50% |== (1)
30-40% |==================== (10)
20-30% |============ (6)
10-20% |== (1)
 0-10% |====================================================== (26)
------------------------------------

 Logical area: RDB$SYSTEM_RECORD for storage area : EMPIDS_LOW
 Larea id: 57,  Record type: 0, Record length: 215, Not Compressed

 Data records: 0, bytes used: 593 (1%)
```

(continued on next page)

Example 4–21 (Cont.) RMU Analyze Areas Command with the Option=Full Qualifier

```
                          used/used+free vs # pages

  >90% | (0)
80-90% | (0)
70-80% | (0)
60-70% | (0)
50-60% |= (1)
40-50% | (0)
30-40% | (0)
20-30% |== (2)
10-20% |= (1)
 0-10% |====================================================== (47)
                        % of max length vs # records

  >90% | (0)
80-90% | (0)
70-80% | (0)
60-70% | (0)
50-60% | (0)
40-50% | (0)
30-40% | (0)
20-30% | (0)
10-20% | (0)
 0-10% |====================================================== (51)

------------------------------------
 Logical area: EMPLOYEES_HASH for storage area : EMPIDS_LOW
 Larea id: 58,  Record type: 0, Record length: 215, Not Compressed

 Data records: 26, bytes used: 989 (2%)
   average length: 38

                          used/used+free vs # pages

  >90% | (0)
80-90% |== (1)
70-80% | (0)
60-70% |== (1)
50-60% |== (1)
40-50% |== (1)
30-40% | (0)
20-30% | (0)
10-20% |===== (2)
 0-10% |====================================================== (20)
```

(continued on next page)

Example 4–21 (Cont.) RMU Analyze Areas Command with the Option=Full Qualifier

```
                         % of max length vs # records

  >90% | (0)
80-90% | (0)
70-80% | (0)
60-70% | (0)
50-60% | (0)
40-50% | (0)
30-40% |== (1)
20-30% |===== (2)
10-20% |=========== (4)
 0-10% |====================================================== (19)

------------------------------------

 Logical area: EMPLOYEES for storage area : EMPIDS_LOW
 Larea id: 63,  Record type: 26, Record length: 121, Compressed

 Data records: 37, bytes used: 2723 (5%)
   average length: 74, compression ratio:  .63


                           used/used+free vs # pages

  >90% |=== (1)
80-90% |======= (2)
70-80% | (0)
60-70% |=== (1)
50-60% | (0)
40-50% | (0)
30-40% |=== (1)
20-30% |======= (2)
10-20% |============== (4)
 0-10% |====================================================== (15)


                         % of max length vs # records

  >90% | (0)
80-90% | (0)
70-80% | (0)
60-70% |========= (5)
50-60% |====================================================== (30)
40-50% |=== (2)
30-40% | (0)
20-30% | (0)
10-20% | (0)
 0-10% | (0)
```

(continued on next page)

Example 4–21 (Cont.) RMU Analyze Areas Command with the Option=Full Qualifier

```
------------------------------------

 Logical area: JOB_HISTORY_HASH for storage area : EMPIDS_LOW
 Larea id: 66,  Record type: 0, Record length: 215, Not Compressed

 Data records: 26, bytes used: 989 (2%)
   average length: 38

                           used/used+free vs # pages

  >90% | (0)
80-90% |== (1)
70-80% | (0)
60-70% |== (1)
50-60% |== (1)
40-50% |== (1)
30-40% | (0)
20-30% | (0)
10-20% |===== (2)
 0-10% |====================================================== (20)

                         % of max length vs # records

  >90% | (0)
80-90% | (0)
70-80% | (0)
60-70% | (0)
50-60% | (0)
40-50% | (0)
30-40% |== (1)
20-30% |===== (2)
10-20% |=========== (4)
 0-10% |====================================================== (19)

------------------------------------

 Logical area: JOB_HISTORY for storage area : EMPIDS_LOW
 Larea id: 69,  Record type: 29, Record length: 42, Compressed

 Data records: 102, bytes used: 3654 (7%)
   average length: 36, compression ratio:  .97
```

(continued on next page)

Example 4–21 (Cont.) RMU Analyze Areas Command with the Option=Full Qualifier

```
                          used/used+free vs # pages

  >90% |======= (1)
80-90% |=============== (2)
70-80% |======= (1)
60-70% |======= (1)
50-60% | (0)
40-50% |=============== (2)
30-40% |=============== (2)
20-30% |======================= (3)
10-20% |====================================================== (7)
 0-10% |====================================================== (7)

                        % of max length vs # records

  >90% | (0)
80-90% |====================================================== (65)
70-80% |============================== (37)
60-70% | (0)
50-60% | (0)
40-50% | (0)
30-40% | (0)
20-30% | (0)
10-20% | (0)
 0-10% | (0)
------------------------------------------------------------------------------
```

The storage area page space use by page histogram (*% used vs # pages*) summarizes the percentage of page space used by data for all logical areas versus the total number of pages in the storage area. The shape of the histogram depends on the type of application and the history of the database. If you have just loaded the database without specifying a PLACEMENT VIA clause in the CREATE STORAGE MAP statement, most pages fall at the top or bottom of the histogram. Oracle Rdb stores data on each database page until the page is full, then proceeds to the next page. When all the data has been stored, some pages are left empty.

In the EMPIDS_LOW storage area, most pages fall at the bottom of the chart. The EMPLOYEES and JOB_HISTORY rows in this storage area are placed using the PLACEMENT VIA INDEX clause that use the EMPLOYEES_HASH and JOB_HISTORY_HASH hashed indexes, respectively.

Further analysis of the logical areas with the aid of the two logical area histograms, *used/used+free vs # pages* and *% of max length vs # of records*, indicates the following:

- The EMPLOYEES_HASH and JOB_HISTORY_HASH logical areas' records are not compressed. The size of the hash bucket in each index averages 38 bytes. Most records (19 of 26 for both EMPLOYEES_HASH and JOB_HISTORY_HASH) are no more than 10 percent of maximum length. The hash buckets are spread out over 26 of 52 pages, and most pages where hash buckets are stored have sufficient free space to allow growth of hash bucket and new records.
- For the EMPLOYEES table, rows are compressed and have an average record length of 74 bytes and an average compression ratio of .63. Most rows (32 of 37) are less than 60 percent of their maximum length. Thirty-seven EMPLOYEES rows are spread out over 26 pages. Most pages (23 of 26 pages) where EMPLOYEES rows are stored have sufficient free space to store additional EMPLOYEES rows.
- For the JOB_HISTORY table, rows are compressed and have an average record length of 36 bytes and an average compression ratio of .97. More than half of the rows (65 of 102) are between 80 and 90 percent of their maximum record length. A total of 102 rows are spread out over 26 pages. Most of the pages (22 of 26 pages) where the JOB_HISTORY rows are stored have sufficient free space to store additional JOB_HISTORY rows.

If the database is frequently updated, the display may show that many pages are nearly 100 percent full. Extensive updates may result in fragmented rows. As this process continues, performance may deteriorate.

When the display of the RMU Analyze Areas command shows that a large number of pages are more than 60 percent full, it may be time to examine database performance. As a general rule, storage areas should be defined with 30 to 50 percent free space.

If database performance is poor and the RMU Analyze Areas display reveals a large number of unused pages or fragmented rows, it may be necessary to use the SQL ALTER DATABASE statement, define new storage areas, specify sets of optimal parameters for the new storage areas, and modify the storage maps to load data from old storage areas to respective new storage areas. See the *Oracle Rdb7 Guide to Database Maintenance* for more information on this procedure.

Using the RMU Analyze Areas Option=Debug Command

When you use the RMU Analyze Areas command and specify the Option=Debug qualifier to gather information on the EMPIDS_LOW storage area, Oracle Rdb displays the output in two parts:

- The first part contains a detailed accounting of the contents of each page in the storage area. This part shows the following in this order:
  - Summary information for each logical area
  - A legend that defines the 11 possible types of records found on a page
  - A detailed accounting of the contents of each storage area page
- The second part contains the same information that displays for the Option=Full qualifier; see the Option=Full qualifier for the RMU Analyze Areas command earlier in this section for information on the format and content of this output.

Example 4–22 shows the information displayed when you use the RMU Analyze Areas command and specify the Option=Debug qualifier.

Example 4–22 RMU Analyze Areas Command with the Option=Debug Qualifier

```
$ RMU/ANALYZE/AREAS=EMPIDS_LOW/OPTION=DEBUG mf_personnel
0------------------------------------------------------------------------------
0
0 Storage-area-name              Area-ID Page-len Last-page File-name
0
0   Logical-area-name              Larea-ID Rec-type    Rec-len Compressed
0------------------------------------------------------------------------------
0
1 EMPIDS_LOW                            2    1024         52 $DUA0:[ORION]EMPIDS_LOW.RDA;1
0
2   RDB$SYSTEM_RECORD                    57       0        215   F
2   EMPLOYEES_HASH                       58       0        215   F
2   EMPLOYEES                            63      26        121   T
```

(continued on next page)

Example 4–22 (Cont.) RMU Analyze Areas Command with the Option=Debug Qualifier

```
2   JOB_HISTORY_HASH                     66       0        215   F
2   JOB_HISTORY                          69      29         42   T
0
0
0------------------------------------------------------------------------------
0
0 TYPES:  0 SPAM              4 B-tree index      8 First data fragment
0         1 AIP               5 Hash index        9 First fragment compressed
0         2 ABM               6 data record      10 Next data fragment
0         3 System record     7 Compressed data  11 Next fragment compressed
0
0 AREA    LAREA   PNO        FREE    SIZE    EXPANDED TYPE TOTAL-SIZE
0------------------------------------------------------------------------------
0
3    2       0          1     948      54          54   0        54
3    2      57          2      62      18          18   3        18
3    2      63          2      62      71         116   7        71
3    2      58          2      62      49          49   5        49
3    2      63          2      62      73         116   7        73
3    2      69          2      62      38          37   7        38
3    2      66          2      62      49          49   5        49
3    2      69          2      62      38          37   7        38
3    2       0          2      62      90          90   5        90
3    2      69          2      62      38          37   7        38
3    2      69          2      62      32          37   7        32
3    2      69          2      62      38          37   7        38
3    2      69          2      62      38          37   7        38
3    2       0          2      62      90          90   5        90
3    2      69          2      62      32          37   7        32
3    2      69          2      62      38          37   7        38
3    2      57          3     964       5           5   3         5
3    2      57          4     964       5           5   3         5
3    2      57          5     344      18          18   3        18
3    2      63          5     344      73         116   7        73
3    2      58          5     344      30          30   5        30
3    2      69          5     344      38          37   7        38
3    2      69          5     344      38          37   7        38
3    2      69          5     344      32          37   7        32
3    2      69          5     344      32          37   7        32
   .
   .
   .
```

The format and content of the output for the RMU Analyze Areas=EMPIDS_LOW command specifying the Option=Debug qualifier are grouped by header as follows (described for the EMPLOYEES_HASH logical area and the first line (PNO 1) in the detailed output section):

- Main header
  - Storage-area-name

    The storage area name (EMPIDS_LOW).

  - Area-ID

    The storage area ID (2).

  - Page-len

    The storage area page length (1024).

  - Last-page

    The last page number in the storage area (52).

  - File-name

    The storage area file name ($DUA0:[ORION]empids_low.rda;1).

- Secondary header (describing only the EMPLOYEES_HASH logical area)

  - Logical-area-name

    The logical area name (EMPLOYEES_HASH).

  - Larea-id

    The logical area ID (58).

  - Rec-type

    The record type (table ID) for the logical area; indexes are always 0 (0).

  - Rec-len

    The maximum size of a record (uncompressed) plus overhead of this type (215). This value is meaningless for logical areas in mixed storage areas.

  - Compressed

    Record compression, a coded value, T=compressed, F=not compressed (F).

- The record types legend

  A legend that denotes the coded value for the 11 possible record types found on a page. This coded value is used in the TYPE column in the detailed page contents section of the output.

- Detailed storage area page contents (described here for the first line, PNO=1)

  - AREA

    Storage area ID number (2).

  - LAREA

  Logical area ID number (0).

  - PNO

    Storage area page number; all logical area IDs found on the page are grouped by page number (1).

  - FREE

    The number of free bytes in the logical area on the specified page (948).

  - SIZE

    The actual record size (in bytes), compressed or uncompressed, in the logical area (54).

  - EXPANDED

    The expanded record size (in bytes) if the record is compressed (54).

  - TYPE

    A coded field for the record type; see the TYPES legend on the output table for an interpretation of the coded value in this column (0, indicating a SPAM page record).

  - TOTAL-SIZE

    The actual total record length, including all record fragments for the record in the logical area (54).

- Option=Full information

  See the information on using the RMU Analyze Areas Option=Full command earlier in this section for a description of this part of the output.

From a scan of the TYPE column in Example 4–22 and a further inspection of the output, you can determine that there are no fragmented rows (TYPE = 8, 9, 10, or 11) on any of the data pages for any of the tables. This is some of the most useful information in the display. A row is fragmented if it grows too large for the amount of available space on its page or if it is larger than a database page when initially stored. If your database has a large number of fragmented rows, then you should consider several possible remedies:

- Use the SQL ALTER DATABASE statement to define new storage areas and specify an optimal set of parameters for the storage areas (such as a larger page size) or lower the SPAM thresholds and modify the storage maps to effect a load of data from the old storage areas to the respective new storage areas. See the *Oracle Rdb7 Guide to Database Maintenance* for more information on this procedure.

- Use the SQL EXPORT and IMPORT statements and create a new database with a larger page size or lower the SPAM thresholds if the page size is large enough.

If several storage areas have fragmentation problems, using the SQL ALTER DATABASE statement may offer the best remedy to the problem.

By inspecting each logical area on a page, you can determine what logical areas are found on the page, the size of the logical area's records, and the amount of free space. For example, for page 2 (PNO 2) in Example 4–22, the four different logical areas plus the system record use 824 bytes of page space, leaving 62 free bytes on the page. This is considered a full page because another parent EMPLOYEES row cannot be stored on the page with its child JOB_HISTORY row. The following records are stored on page 2:

- Two EMPLOYEES rows (116 bytes each, total 232 bytes)
- Eight JOB_HISTORY rows (37 bytes each, total 296 bytes)
- One SYSTEM record (18 bytes)
- Two hash buckets (49 bytes each, total 98 bytes)
- Two duplicate node records (90 bytes each, total 180 bytes)

By analyzing the page contents in this way, you can get a good idea of how space is being utilized on pages in a storage area. This information coupled with the histogram output can give you an early indication of potential problems. You should perform analysis on your database on a regular basis, especially if your application primarily updates the database, which is continuously growing. If you know the page range desired, you can use the Start and End qualifiers to analyze just this desired range of pages.

You can also use the output from the RMU Analyze Area command specifying the Option=Debug qualifier as input to a program of your own design to devise your own summary reports. For example, you can summarize information by logical area and by page.

4.2.1.2 RMU Analyze Lareas Display

This section describes the format and content of the output when you use the RMU Analyze command with the Lareas and Option qualifiers.

Using the RMU Analyze Lareas Option=Normal Command

When you use the RMU Analyze command and specify the Lareas and Option=Normal qualifiers, Oracle Rdb displays a summary of important information for *all* logical areas defined within each storage area. The information that is displayed is identical in format and content to that produced when you use the RMU Analyze command and specify the Areas and Option=Normal qualifiers. See Using the RMU Analyze Areas Option=Normal Command in Section 4.2.1.1 for a description of this information.

A practical use for this command is to selectively display specific logical areas within a specific storage area by specifying the Areas qualifier along with the Lareas qualifier. Note that the logical areas with the same name and logical ID number within the RDB$SYSTEM storage area do not display if they are not specified. When you use the RMU Analyze command and specify the Areas=DEPARTMENTS, Lareas=(DEPARTMENTS, DEPARTMENTS_INDEX), and Option=Normal qualifiers, the information for just these areas is displayed, as shown in Example 4–23.

Example 4–23 RMU Analyze Areas Command with the Lareas and Option=Normal Qualifiers

```
$ RMU/ANALYZE/AREAS=DEPARTMENTS /LAREAS=(DEPARTMENTS,DEPARTMENTS_INDEX) -
_$ /OPTION=NORMAL mf_personnel

-----------------------------------------------------------------------------

 Storage analysis for storage area: DEPARTMENTS - file: $DUA0:[ORION]DEPARTMENTS.RDA;1
 Area_id: 5,  Page length: 1024,  Last page: 28

 Bytes free: 25144 (88%), bytes overhead: 2062 (7%)
 Spam count: 1, AIP count: 0, ABM count: 0
 Data records: 27, bytes used: 1466 (5%)
   average length: 54, compression ratio:  .85
   index records: 1, bytes used: 428 (1%)
     B-Tree: 428, Hash: 0, Duplicate: 0, Overflow: 0

------------------------------------

 Logical area: DEPARTMENTS_INDEX for storage area : DEPARTMENTS
 Larea id: 72,  Record type: 0, Record length: 215, Not Compressed

 Data records: 1, bytes used: 428 (1%)
   average length: 428
------------------------------------
```

(continued on next page)

Example 4–23 (Cont.) RMU Analyze Areas Command with the Lareas and Option=Normal Qualifiers

```
Logical area: DEPARTMENTS for storage area : DEPARTMENTS
Larea id: 73,  Record type: 28, Record length: 55, Compressed

Data records: 26, bytes used: 1038 (4%)
  average length: 40, compression ratio:  .80
```

Note that the DEPARTMENTS logical area, even though it also exists within the RDB$SYSTEM storage area and is empty, is not displayed because this storage area was not specified.

Using the RMU Analyze Lareas Option=Full Command

When you use the RMU Analyze Lareas command and specify the Option=Full qualifier, the same information displays that displays for the RMU Analyze Lareas command with the Option=Normal qualifier. In addition, the three types of histograms described in Section 4.2.1.1 for the RMU Analyze command using the Areas and Option=Full qualifiers are also displayed. Refer to this section for a description of the histograms.

Using the Lareas and Option=Full qualifiers displays the following information, in this order:

- Summary information and a summary histogram for *all* logical areas by storage area
- Specific information for each logical area and the storage area page space use by page histogram
- A pair of histograms for each logical area within each storage area

Using the RMU Analyze Lareas Option=Debug Command

When you use the RMU Analyze Lareas command and specify the Option=Debug qualifier, the output forms two parts as described for the RMU Analyze Areas Option=Debug command in Section 4.2.1.1. However, only the information for the selected logical areas and the system records (logical area ID 0) is shown in the output:

- The first part contains a detailed accounting of the contents of each page in the storage area. This part shows the following in this order:
  - Summary information for each logical area

- A legend that defines the 11 possible types of records found on a page
- A detailed accounting of the contents of each storage area page

- The second part contains the same information that displays for the Option=Full qualifier; see the Option=Full qualifier for the RMU Analyze Areas command in Section 4.2.1.1 for information on the format and content of this output.

Refer to the descriptions of the RMU Analyze Areas command with the Option=Debug qualifier in Section 4.2.1.1 for information and an example of this display.

4.2.1.3 Performance Monitor Storage Area Information Screen

Oracle Rdb allows you to display information for each storage area in the database. Select the Storage Area Information option from the Database Parameter Information submenu. The following is an example of a Storage Area Information screen:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1996 13:57:59
Rate: 3.00 Seconds          Storage Area Information        Elapsed: 03:37:05.72
Page: 1 of 60      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------

Storage area "RDB$SYSTEM"
Area ID number is 1
Filename is "RDBVMS_USER1:[LOGAN.V70]MF_PERS_DEFAULT.RDA;1"
Access mode is read/write
Page format is uniform
Page size is 2 blocks
- Current physical page count is 1036
Row level locking is enabled
Row caching is disabled
No row cache is defined for this area
Extends are enabled
- Extend area by 20%, minimum of 99 pages, maximum of 9999 pages
- Area has been extended 8 times
Volume set spreading is enabled
Snapshot area ID number is 31
SPAMs are enabled
- Interval is 1089 data pages
--------------------------------------------------------------------------------
Exit Help Menu >next_page <prev_page Options Refresh Set_rate Write !
```

See the Performance Monitor help for information about this screen.

4.2.1.4 Performance Monitor I/O Statistics Screen

Oracle Rdb allows you to display I/O statistics for each file in the database. When you select IO Statistics (by file) in the Performance Monitor, Oracle Rdb displays a menu of the files that comprise the database and for which you can view statistics. The following is an example of an I/O Statistics (by file) menu:

```
Node: ALPHA3                                               14-MAR-1995 14:18:24
Rate: 3.00 Seconds               Select File               Elapsed: 00:05:54.89
Page: 1 of 1    RDBVMS +------------------------------+  .RDB;1    Mode: Online
---------------------- |  A. File IO Overview         | ----------------------
statistic.........     |  B. Device IO Overview       | ....... average......
name..............     |  C. Device Information       | ....... per.trans....
transactions           |  D. root file                |      0          0.0
verb successes         |  E. AIJ file                 |      0          0.0
verb failures          |  F. RUJ file                 |      0          0.0
                       |  G. ACE file                 |
synch data reads       |  H. all data/snap files      |      0          0.0
synch data writes      |  I. data file MF_PERS_DEFAULT |      0          0.0
asynch data reads      |  J. data file EMPIDS_LOW     |      0          0.0
asynch data writes     |  K. data file EMPIDS_MID     |      0          0.0
RUJ file reads         |  L. data file EMPIDS_OVER    |      0          0.0
RUJ file writes        |  M. data file DEPARTMENTS    |      0          0.0
AIJ file reads         |  N. data file SALARY_HISTORY |      0          0.0
AIJ file writes        |  O. data file JOBS           |      0          0.0
ACE file reads         |  P. data file EMP_INFO       |      0          0.0
ACE file writes        |  Q. <<more>>                 |      0          0.0
root file reads        |                              |     17          0.0
root file writes       +------------------------------+      0          0.0

--------------------------------------------------------------------------------
Type <return> or <letter> to select file, <control-Z> to cancel menu
```

Note that the display includes all .rda and .snp files. You cannot use this screen in graph format. Note that the information applies from the time that your Performance Monitor session began, or since the accumulators were last reset using the Reset option.

For information about each of the fields shown in this display, see the Performance Monitor help.

The following example shows the File I/O Statistics screen for all data/snap files:

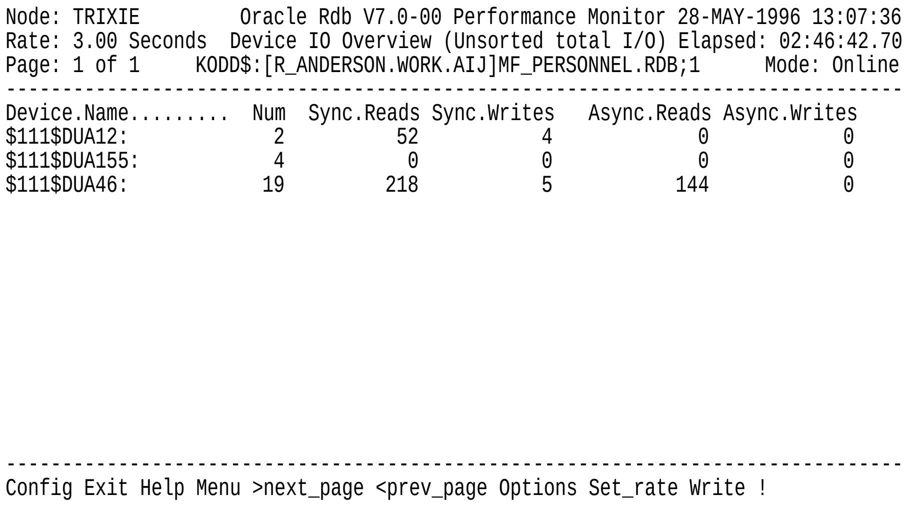
```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1996 12:52:53
Rate: 3.00 Seconds            File IO Statistics          Elapsed: 02:31:59.67
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
                         For File: All data/snap files
statistic.......... rate.per.second............ total....... average......
name............... max..... cur..... avg....... count....... per.trans....
total I/Os                       0                     36          12.0
    (Synch. reads)        0       0       0.0          36          12.0
    (Synch. writes)       0       0       0.0           0           0.0
    (Extends)             0       0       0.0           0           0.0
    (Asynch. reads)       0       0       0.0           0           0.0
    (Asynch. writes)      0       0       0.0           0           0.0

statistic.......... blocks.transferred.........  stall.time.(x100)...........
name............... avg.per.I/O.. total........  avg.per.I/O... total........
total I/Os                   6.0          216            0.6           25
    (Synch. reads)           6.0          216            0.6           25
    (Synch. writes)          0.0            0            0.0            0
    (Extends)                0.0            0            0.0            0
    (Asynch. reads)          0.0            0            0.0            0
    (Asynch. writes)         0.0            0            0.0            0
--------------------------------------------------------------------------------
Exit Help Menu Options Reset Set_rate Unreset Write !
```

OpenVMS VAX OpenVMS Alpha The Device IO Overview screen displays the synchronous and asynchronous read and write I/O counts for all devices that contain live or snapshot storage areas, or the database root file.

The Device IO Overview screen does not display information for devices on which .ruj, .aij, or .ace files reside.

Storage areas that are added to or deleted from the database will be automatically reflected in the screen.

The following example shows a Device IO Overview screen:



```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1996 13:07:36
Rate: 3.00 Seconds  Device IO Overview (Unsorted total I/O) Elapsed: 02:46:42.70
Page: 1 of 1     KODD$:[R_ANDERSON.WORK.AIJ]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
Device.Name.........  Num  Sync.Reads Sync.Writes   Async.Reads Async.Writes
$111$DUA12:             2          52           4             0            0
$111$DUA155:            4           0           0             0            0
$111$DUA46:            19         218           5           144            0








--------------------------------------------------------------------------------
Config Exit Help Menu >next_page <prev_page Options Set_rate Write !
```

The “Device.Name” column displays the expanded storage area device name; concealed logicals are expanded to eliminate duplicated entries.

The “Num” column displays the number of live and snapshot storage areas (and possibly the root file) included in the device information.

The remaining columns display information based on the selected configuration. The Device IO Overview screen name in the header region indicates which display configuration has been selected.

See the Performance Monitor help for information on the Device IO Overview configuration options. ♦

OpenVMS VAX OpenVMS Alpha

You can use the Device Information screen to determine when a storage area device is low on disk space. The following example shows a Device Information screen:

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1996 13:10:46
Rate: 3.00 Seconds             Device Information          Elapsed: 02:49:52.38
Page: 1 of 1     USER$:[ORACLEUSER.WORK.AIJ]MF_PERSONNEL.RDB;1    Mode:   Online
--------------------------------------------------------------------------------
Device.Name......... Status. #Err Volume.Label FreeBlocks Max#Blocks %Full
$111$DUA46:          Mounted    0 KODA_USER2       377727    2376153  84.1
$111$DUA12:          Mounted    0 KODA_TEST1       595920    1216665  51.0
$111$DUA13:          Mounted    0 KODA_TEST2         6350    1216665  99.4







--------------------------------------------------------------------------------
Exit Help Menu >next_page <prev_page Options Set_rate Write !
```

The Device Information screen provides an online view of the storage area device information local to a particular database.

See the Performance Monitor help for a complete description of this screen. ♦

4.2.2 Page Size

The page size value determines the number of 512-byte blocks Oracle Rdb uses for the storage page of the database. The default value is 2 blocks per page, or a total of 1024 bytes.

If your row size is larger than 950 bytes, or you have segmented strings that are larger than 950 bytes, you need a larger page size. You might also want to increase the page size value if you have a bill of materials (recursive relationship) application.

It is important to remember that, in addition to user data and free space, each database page stores such items as:

- Page header

  A fixed-length portion of the page that contains information about the page, including the number of bytes available.

- Line index

  A directory to all the storage segments on the page.

- Record version number

> Because of its relational capabilities, Oracle Rdb allows for dynamic record type modification. The user always views the record as the current definition of the record type. To keep track of the record type versions, each user record contains a version number field.

Each page, therefore, makes fewer bytes available for user data than the actual page size value specified for the PAGE SIZE parameter indicates. The overhead for each page is estimated to be about 7 to 10 percent. The amount of overhead depends on the size of the database page. The larger the page, the less overhead space Oracle Rdb requires to maintain information about the contents of the page. Therefore, when you calculate the row sizes in the database, allow enough space in the database page size for the largest row as well as for the overhead.

For retrieval, the database operates most efficiently when the database page is nearly full. For this reason, try to choose a page size that accommodates the size of the rows in your tables. If the page size is too large, you waste space. Sequential I/O operations benefit from large pages, while random I/O operations benefit from small pages. However, if the page size is too small, Oracle Rdb may split (fragment) rows across more than one database page.

As you fill a database page, you use up free space until space is no longer available for a complete row. If you then modify a row and make it larger, Oracle Rdb may need to fragment the row. A fragmented row is divided across the available free space. Pointers attached to the row's fragments indicate the location of other fragments. However, retrieving fragmented rows is more costly than retrieving whole rows. For this reason, you should try to avoid fragmentation when you design and maintain your database.

A common use for the RMU Dump command is to track down problems with storage allocation and record placement by dumping the SPAM pages, and then the individual data pages. You can use the Spams_Only qualifier of the RMU Dump command to dump only the SPAM pages in the selected storage areas and page range. See the *Oracle RMU Reference Manual* for more information on using the RMU Dump command.

4.2.3 Allocation Size

When you specify the initial database space with the ALLOCATION parameter, Oracle Rdb reserves disk space for the database file based on the PAGE SIZE value you include in the SQL CREATE DATABASE statement for each storage area you define with the SQL CREATE STORAGE AREA statement. The allocation size is the amount of space used to store the rows or records in each storage area when it is created. Note that pages are allocated in groups of three and rounded up to the nearest increment of three pages. For example, if you specify an allocation of 25 pages, the storage area will actually be allocated

27 pages. The default value of ALLOCATION is 400 pages. If you use the default page size, the initial allocation size of the storage area file is 800 512-byte disk blocks. If you know how large each storage area will be, or if you are using the SQL EXPORT statement, set the allocation to the final size of each storage area. This causes Oracle Rdb to allocate space contiguously where possible and results in better system performance for large databases.

4.2.4 Page Format

The default storage area page format is uniform. This page format is suitable for everything except special cases such as parent-child relationships with exact match retrievals. In other words, use a uniform page format unless some special case dictates otherwise. This is generally the case for many applications with fewer than 10 to 15 tables that are small-to-medium-sized single-file databases. The same is true for multifile databases where some tables are stored in the default RDB$SYSTEM storage area, while others are stored individually in separate storage areas. These types of applications would probably use sorted indexes for accessing rows in moderately sized tables, and sequential access methods for accessing rows in small tables. For some applications in this category with one or several large tables, access to rows might be improved with a hashed index defined in a storage area with a mixed page format.

In applications with more than 15 tables that are medium-to-very-large complex databases, a uniform page format is again suitable for everything except special cases such as parent-child relationships with exact-match retrievals. While large, complex databases may benefit from judicious use of mixed page format, there is not a one-to-one relationship between database size, complexity, and storage area type. Some of the benefits include the use of hashed indexes, optimizing SPAM intervals, controlling SPAM thresholds, and storing parent-child rows together in the same mixed storage area. The benefits of selecting SPAM thresholds and optimizing SPAM intervals are described in more detail in Section 4.2.5 and Section 4.2.6, while the benefits of using hashed indexes with parent-child tables are described in Section 3.9.7.

4.2.5 General Guidelines for Selecting SPAM Threshold Values

A SPAM page manages the free space on a range of data pages by means of a 2-bit entry for each data page. The 2-bit entry indicates the threshold value for a particular page. You can set three threshold values, which indicate the relative fullness of a page for pages in storage areas with a mixed page format, or for pages in logical areas with a uniform page format.

Thresholds are calculated against the maximum amount of free space on a page. To determine the maximum amount of free space on a page, subtract the page overhead from the page size. For example, a page in the DEPARTMENTS storage area in the mf_personnel database is 1024 bytes and has 60 bytes of page overhead, which means the maximum amount of free space is 964 bytes.

Refer to the *Oracle Rdb7 Guide to Database Maintenance* for more information on SPAM pages and thresholds.

4.2.5.1 Thresholds for Mixed Format Pages

For multifile databases with mixed page format storage areas, you can set three threshold values associated with the SPAM page free space inventory lists. The default values are 70, 85, and 95 percent. These represent ranges of guaranteed free space on each data page as 30, 15, and 5 percent, respectively.

When a table has a storage map that specifies placement via index and the target page is full, the pages in the buffer are searched for free space. If a page in the buffer has sufficient space to store the record, the record is stored. If no space is found among the pages in the buffer, then Oracle Rdb scans the SPAM pages in the storage area for sufficient space to store the record. Searching the pages in the buffer for free space before scanning the SPAM pages can improve the performance of storing records by reducing the I/O operations needed to store them.

Oracle Rdb uses a storage algorithm that guarantees free space when using SPAM thresholds and a placement index. With this algorithm, Oracle Rdb never stores a new record on a page that has reached the third threshold value. Thus, a table that uses the PLACEMENT VIA INDEX clause with a hashed index no longer writes directly to the target page without first verifying that the page has not reached the third threshold value.

By knowing the length of the rows being stored in an area and the size of each database page, you can select threshold values that speed up row storage. The goal is to minimize the amount of searching necessary to store a complete row on one database page. Threshold values should be based on the size and storage frequency of the rows being stored in the database. Follow these general guidelines to select SPAM threshold values:

- Set threshold values that guarantee a *typical* row (in terms of size and frequency of storage) can be stored at least one more time on a single database page.
- Use the THRESHOLDS ARE option for each storage area if the length of the rows varies greatly from one storage area to another.

- Within each storage area, if you are storing multiple tables (EMPLOYEES and JOB_HISTORY, for instance) of different sizes (a common occurrence), do the following:
  - Set the lowest threshold value so that it guarantees any pages that have not reached this fullness percentage can still store the largest record type at least once. You may have to increase the number of blocks per page to accommodate this arrangement. To increase the blocks per page, use the PAGE SIZE IS page-blocks BLOCKS option in the SQL CREATE DATABASE or SQL IMPORT statement.
  - Set the middle threshold value so that the pages that have not reached this fullness percentage can still store at least one more of the smaller record types (an index or table row).
  - Oracle Rdb never stores a row on a page at the highest threshold. You can use the value set for this threshold to reserve free space on a page for growth due to row updates.

The third threshold value determines at what percentage you consider the database page to be completely full. If the third threshold value is set at 85, no rows are stored on database pages with less than 15 percent free space remaining. Most importantly, Oracle Rdb does not waste time examining any database pages that have reached the third fullness threshold.

Note

The SPAM algorithm assumes that data compression will not shrink the row being stored. Therefore, when you think about the size of your largest rows, use the row's uncompressed size.

You can determine the number of bytes for each threshold in a storage area as follows:

```
SPAM_THRESHOLD_n = THRESHOLD_n * (maximum free space - 1) /100
```

Example 4–24 shows the number of bytes for each threshold for the DEPARTMENTS storage area when the default thresholds of 70, 85, and 95 percent are used.

Example 4–24 Determining the Size of Thresholds for the DEPARTMENTS Storage Area

```
SPAM_THRESHOLD_1 = 70 * 963 / 100 = 674

SPAM_THRESHOLD_2 = 85 * 963 / 100 = 818

SPAM_THRESHOLD_1 = 95 * 963 / 100 = 914
```

4.2.5.2 Thresholds for Uniform Format Pages

You can specify three threshold values for a logical area when you create or change a storage map or index. By taking into account data compression or index node size when you set threshold values, you can maximize the number of rows stored on a page. Oracle Rdb is aware of the compressed row or index node size. In the previous example, you could set the third threshold value at 94 percent, indicating a fullness threshold of 893 bytes (950 * .94), leaving 57 bytes of free space. This would guarantee pages in this logical area would store the maximum number of 60-byte rows. Refer to the *Oracle Rdb7 Guide to Database Maintenance* for information on how SPAM pages work.

You can set threshold values for a logical area by using the SQL CREATE or ALTER INDEX statement or the CREATE or ALTER STORAGE MAP statement.

The following examples illustrate how to set thresholds for logical areas. Refer to the *Oracle Rdb7 SQL Reference Manual* for more information on SQL syntax.

```
SQL> CREATE INDEX index_name
cont>   ON table_name (column_name)
cont>   TYPE IS SORTED
cont>   STORE USING (column_name)
cont>      IN area_name (THRESHOLDS ARE (value1,value2,value3))
cont>      WITH LIMIT OF (literal)
cont>      IN area_name (THRESHOLDS ARE (value1,value2,value3))
cont>      WITH LIMIT OF (literal)
cont>      OTHERWISE IN area_name (THRESHOLDS ARE (value1,value2,value3));
```

The preceding example defines a sorted index and uses the store clause to partition the index into three different storage areas. The THRESHOLDS ARE clauses can establish up to three threshold values for each storage area.

```
SQL> CREATE STORAGE MAP map_name FOR table_name
cont>   STORE USING (column_name)
cont>      IN area_name (THRESHOLDS ARE (value1,value2,value3))
cont>      WITH LIMIT OF (literal)
cont>      IN area_name (THRESHOLDS ARE (value1,value2,value3))
cont>      WITH LIMIT OF (literal)
cont>      OTHERWISE IN area_name (THRESHOLDS ARE (value1,value2,value3))
cont>   ENABLE COMPRESSION;
```

The preceding example creates a storage map, sets the logical area thresholds for three specified, partitioned areas, and enables compression. Note that if you specify THRESHOLDS ARE (0,0,0) or if you do not specify threshold values for an area, Oracle Rdb uses the default storage behavior. That is, a page is marked as full or not full, depending on whether the page can store at least one more row.

```
SQL> CREATE STORAGE MAP map_name FOR table_name
cont>   STORE IN area_name
cont>   ENABLE COMPRESSION
cont>   THRESHOLDS ARE (value1,value2,value3);
```

The preceding example creates a storage map that stores all the rows from a single table in one area and sets the logical area thresholds.

```
SQL> ALTER STORAGE MAP map_name
cont>   STORE IN area_name
cont>   ENABLE COMPRESSION
cont>   THRESHOLDS ARE (value1,value2,value3)
cont>   PLACEMENT VIA INDEX index_name;
```

The preceding example uses the PLACEMENT VIA INDEX clause to store rows in a storage area. The threshold values established apply only to areas added during this CREATE STORAGE MAP statement. Areas subsequently added by an ALTER STORAGE MAP statement default to (0,0,0) unless you specify an explicit THRESHOLDS ARE clause.

To determine the threshold values for a logical area, you need to calculate the free space available on a database page and the compression ratio of rows that will be stored on the page. Refer to the *Oracle Rdb7 Guide to Database Design and Definition* for information on calculating page size; refer to Section 4.3.3 for information on the data compression option.

You can use the SQL SHOW STORAGE MAP and SHOW INDEX statements to display the original text used to create a storage map or index. You can use the RMU Dump Larea=RDB$AIP or RMU Extract Item=(Index,Storage) commands to display current information about thresholds.

4.2.6 Optimizing SPAM Intervals

Accepting the default value of 216 pages for the SPAM interval is fine when your database is medium-sized (smaller than 100 Mb or 200,000 blocks), for example, and is relatively simple (containing only 10 to 15 tables). Generally, for a large database (between 100 Mb and 1 Gb in size) that contains only 10 to 15 tables, a few storage areas might be quite large (between 10 Mb or 20,000 blocks and 100 Mb or 200,000 blocks). When you know that the storage areas are going to range between 10 to 100 Mb, you should consider adjusting specific storage area parameters. In this case, select a SPAM interval value that matches your application's activity (insert-intensive as opposed to update-intensive activity) for that storage area. A 10 Mb storage area with a page size of 2 blocks has 10,000 pages and, with a default SPAM interval of 216 pages, provides about 46 SPAM pages for the storage area. A 100 Mb storage area can have about 460 SPAM pages or potentially many more I/O operations to find free space for records.

A higher interval value can result in greater contention for SPAM pages in an update-intensive environment. That is, with a single SPAM page that contains information about the amount of free space on many data pages, some users who attempt to update rows in the range of data pages maintained by that SPAM page may have to wait until other users complete their use of the free space inventory list on that SPAM page.

A summary of the INTERVAL IS trade-offs is presented in the following list:

- A larger interval value for a storage area may help reduce disk I/O operations when Oracle Rdb is trying to locate free space for an insert-intensive application. But a larger interval value may also increase SPAM page locking when many simultaneous update users are accessing this portion of the storage area for an update-intensive application.
- With smaller interval values, you can reduce SPAM page-locking problems for update-intensive environments. However, smaller interval values can increase the number of disk I/O operations (and elapsed time) required to locate free row space for an insert-intensive application.

Determine whether disk I/O operations or SPAM page locking costs your applications more time, and adjust the INTERVAL IS value accordingly. To determine the disk I/O operations, use the IO Statistics (by file) screen in the Performance Monitor. See Section 2.2 for information on invoking and interpreting the Performance Monitor screens.

See the *Oracle Rdb7 SQL Reference Manual* for syntax and arguments for setting interval and threshold values. See the *Oracle Rdb7 Guide to Database Maintenance* for more information on SPAM thresholds, for additional examples on setting interval and threshold values, and for details on what the threshold values mean and how you can use them to your application's advantage.

4.2.7 Placing Snapshot, Storage Area, and Database Files on Separate Disks

You can reduce disk I/O contention in certain Oracle Rdb applications by placing the snapshot (.snp) file and the database root (.rdb) file on separate disk devices for single-file databases. For multifile databases, you can reduce disk I/O contention by placing storage area (.rda) files on different disks from the .rdb and .snp files. You should place .ruj and .aij files on disks where disk contention can be minimized and database recovery can be carried out most efficiently when needed. Be sure that .aij files are not placed on disks with other database files.

Placing .snp files apart from .rda files for the same storage area improves performance in applications that have simultaneous read-only and read/write transactions. It also improves performance for applications that have many read/write transactions when snapshots are enabled and the default IMMEDIATE option is used because the I/O operations are spread over two disks. Also, because disk write operations at commit time are done asynchronously, response time improves by spreading the I/O operations over two or more disks.

Remember that the location of the .snp file for each .rda file is specified in the SQL CREATE DATABASE statement with the CREATE STORAGE AREA clause. You cannot change the .snp file name using the ALTER STORAGE AREA clause of the ALTER DATABASE statement.

4.2.8 Initializing, Moving, and Changing the Allocation of Snapshot Files

The RMU Repair command can be used to initialize, move, and change the allocation of .snp files. These operations may be necessary when a disk with a snapshot file has a hardware problem or is removed in a hardware upgrade, or when a snapshot file has grown too large and you want to truncate it. Oracle Corporation recommends that you use the RMU Backup command to perform a full backup of your database before using the RMU Repair command on the database.

You can initialize the snapshot files for specific storage areas by using the RMU Repair Nospams Initialize=Snapshots command with the Area qualifier. Example 4–25 shows how to initialize departments.snp and jobs.snp for the mf_personnel database.

Example 4–25 Using RMU Repair to Initialize Existing Snapshot Files for Specific Storage Areas in a Database

```
$ RMU/REPAIR/NOSPAMS/INITIALIZE=SNAPSHOTS/AREA=(DEPARTMENTS,JOBS) mf_personnel
```

```
$ rmu -repair -nospams -initialize=snapshots
> -area=(DEPARTMENTS,JOBS) mf_personnel
```

By using the RMU Repair Nospams Initialize=Snapshots command without the Area qualifier, you can initialize all the snapshot files for the mf_personnel database, as shown in Example 4–26.

Example 4–26 Using RMU Repair to Initialize All the Existing Snapshot Files for a Database

```
$ RMU/REPAIR/NOSPAMS/INITIALIZE=SNAPSHOTS mf_personnel
```

```
$ rmu -repair -nospams -initialize=snapshots mf_personnel
```

When you specify the CONFIRM keyword with the Initialize=Snapshots qualifier, you can use RMU Repair to not only initialize, but also to optionally rename, move, and change the allocation of snapshot files. The CONFIRM keyword causes Oracle Rdb to prompt you for a name and allocation for one or more snapshot files. If you use the Area qualifier, you can select the snapshot files in the database that you want to modify. If you omit the Area qualifier, all the snapshot files for the database will be initialized and Oracle Rdb will prompt you interactively for an alternative file name and allocation for each snapshot file. By specifying a new file name for a snapshot file, you can change the location of the snapshot file. By specifying a new allocation for a snapshot file, you can truncate a snapshot file or make it larger.

OpenVMS VAX OpenVMS Alpha

In Example 4–27, the RMU Repair command is used to initialize and rename departments.snp, to initialize and move salary_history.snp, and to initialize, move, and truncate jobs.snp.

Example 4–27 Using RMU Repair to Initialize, Rename, Move, and Truncate Snapshot Files for Specific Storage Areas in a Database

```
$ RMU/REPAIR/NOSPAMS/INITIALIZE=SNAPSHOTS=CONFIRM -
_$ /AREA=(DEPARTMENTS,JOBS,SALARY_HISTORY) mf_personnel
%RMU-I-FULBACREQ, A full backup of this database should be performed after
RMU/REPAIR
```

(continued on next page)

Example 4–27 (Cont.) Using RMU Repair to Initialize, Rename, Move, and Truncate Snapshot Files for Specific Storage Areas in a Database

```
Area DEPARTMENTS snapshot filename [SQL1:[TEST]DEPARTMENTS.SNP;1]: NEW_DEPT
Area DEPARTMENTS snapshot file allocation [10]?
Area SALARY_HISTORY snapshot filename [SQL1:[TEST]SALARY_HISTORY.SNP;1]: SQL2:
Area SALARY_HISTORY snapshot file allocation [10]?
Area JOBS snapshot filename [SQL1:[TEST]JOBS.SNP;1]: SQL2:[TEST2]
Area JOBS snapshot file allocation [10]? 5
$
```

♦

Use the Performance Monitor Storage Area Information screen located in the Database Parameter Information submenu to display the new location and new allocation of a snapshot file.

You should be careful when specifying names for new snapshot files with RMU Repair. If you specify the name of a file that already exists and was created for the database, it will be initialized as you requested.

If you mistakenly initialize a live database file in this way, you should not use the database until the error has been corrected. Use the RMU Restore command to restore the database to the condition it was in when you backed it up just prior to issuing the RMU Repair command. If you did not back up the database before issuing the RMU Repair command, you will have to restore from your most recent backup file and then recover from .aij files (if the database had after-image journaling enabled).

If you specified the wrong snapshot file (for example, if you specified JOBS.SNP for all the snapshot file name requests in Example 4–27), you can correct this by issuing the RMU Repair command again with the correct snapshot file names.

After the RMU Repair operation completes, delete old snapshot files and use the RMU Backup command to perform a full backup of your database.

You can truncate snapshot files on line. This means the database can continue to be available to users during the snapshot file truncation.

Example 4–28 shows how to truncate a database or storage area snapshot file on line.

Example 4–28 Truncating a Database or Storage Area Snapshot File On Line

```
SQL> -- Truncate the database snapshot file on line:
SQL> ALTER DATABASE FILENAME mf_personnel
cont>   SNAPSHOT ALLOCATION IS 18 PAGES;
SQL> --
SQL> -- Truncate a storage area snapshot file on line:
SQL> ALTER DATABASE FILENAME mf_personnel
cont>   ALTER STORAGE AREA EMPIDS_LOW
cont>      SNAPSHOT ALLOCATION IS 6 PAGES;
```

When an online snapshot truncation operation is started, Oracle Rdb waits for a read-only transaction quiet point and then temporarily adopts a deferred snapshot mode for the database. When the deferred snapshot mode is in effect, Oracle Rdb truncates the snapshot file to the requested size. Read-only transactions are blocked when the truncation is taking place. After the snapshot file has been truncated, Oracle Rdb restores the database to the previous snapshot mode (the deferred snapshot mode is in effect only while the snapshot file is being truncated).

During online snapshot truncation, users acquire two new locks when they attach to the database, resulting in a slight increase in the time it takes them to attach to the database.

Truncating snapshot files is recommended in the following situations:

- After long read-only transactions
- To prevent the performance degradation caused by large, fragmented snapshot files

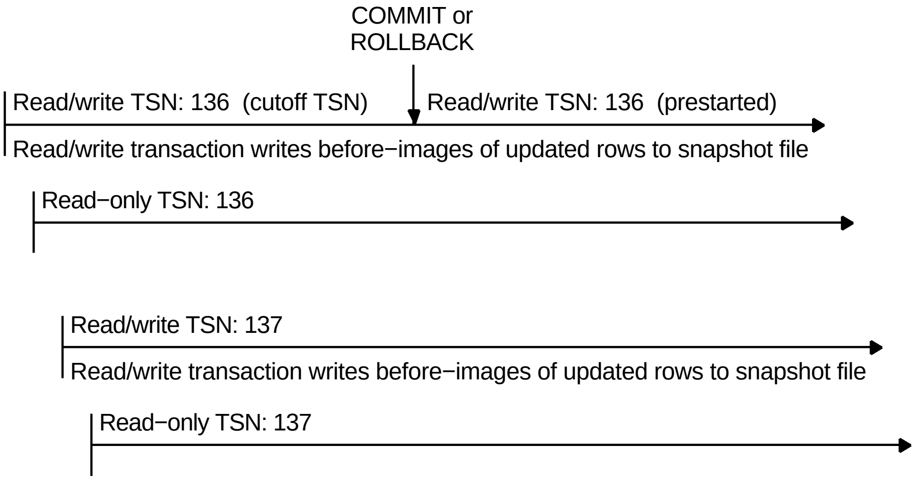
4.2.9 Snapshot File Growth and Prestarted Transactions

A prestarted transaction is an optimization that Oracle Rdb uses to reduce I/O when a transaction starts. With prestarted transactions enabled (the default), when a process issues a COMMIT or ROLLBACK statement to end a read/write transaction, Oracle Rdb immediately starts a new read/write transaction for that process (the new read/write transaction that is immediately started is called a **prestarted transaction**). Oracle Rdb gives the new prestarted transaction the same transaction sequence number (TSN) as the original read/write transaction. Prestarted transactions save I/O because Oracle Rdb does not have to reserve a new TSN from the database root file.

Oracle Rdb also lets you disable prestarted transactions. You might want to disable prestarted transactions when they are causing excessive snapshot file growth in a database. More I/O to the root file is incurred for new transactions when you disable prestarted transactions.

Figure 4–11 shows how Oracle Rdb handles the writing of snapshot records when prestarted transactions are enabled. In Figure 4–11, four transactions are accessing a database. The first transaction is a read/write transaction, the second a read-only transaction, the third a read/write transaction, and the fourth a read-only transaction. Oracle Rdb assigns a "cutoff TSN" to the oldest read/write transaction in the database. If the TSN for a record on a snapshot page is less than the cutoff TSN, then Oracle Rdb can reclaim the record.

Figure 4–11 Prestarted Transactions and Snapshot File Growth


COMMIT or ROLLBACK
Read/write TSN: 136 (cutoff TSN)
Read/write TSN: 136 (prestarted)
Read/write transaction writes before−images of updated rows to snapshot file
Read−only TSN: 136
Read/write TSN: 137
Read/write transaction writes before−images of updated rows to snapshot file
Read−only TSN: 137

NU−2766A−RA

Because prestarted transactions are enabled in Figure 4–11, when the first read/write transaction (TSN 136, the cutoff TSN) commits or rolls back, a new read/write transaction is prestarted. Oracle Rdb gives TSN 136 to the prestarted transaction, so TSN 136 remains the cutoff TSN. Because the first read/write transaction (TSN 136) is still the oldest read/write transaction in the database, Oracle Rdb cannot reclaim the snapshot records written by the cutoff TSN's process and by the other read/write transaction (TSN 137). The snapshot files will continue to grow until one of the following occurs:

- The cutoff TSN's process detaches from the database or is terminated.
- The cutoff TSN's process performs a commit or rollback operation, then starts an explicit read-only transaction that is rolled back immediately:

```
SQL> COMMIT;
SQL> SET TRANSACTION READ ONLY;
SQL> ROLLBACK;
```

Usually, having prestarted transactions enabled does not cause snapshot files to grow excessively. However, if your application uses a server that is attached to the database for long periods of time, the cutoff TSN may be very old. In this situation, the snapshot files grow large because Oracle Rdb can reclaim fewer snapshot records.

In Example 4–29, output from the RMU Dump Users command provides more information on the prestarted transactions scenario shown in Figure 4–11. Example 4–29 shows the two read/write transactions and two read-only transactions in Figure 4–11. Prestarted transactions are used for the read/write transactions. Note that after the first read/write transaction (TSN 136, the cutoff TSN) is committed or rolled back, Oracle Rdb begins a prestarted transaction with the same TSN, so this transaction remains the cutoff TSN.

Example 4–29 Displaying Transactions When Prestarted Transactions Are Enabled

```
$! While the four transactions are in progress:
$ RMU/DUMP/USERS mf_personnel
Active user with process ID 71A01142
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 19
    Recovery journal filename is "SQL_USER1:[RDM$RUJ]MF_PERSONNEL$009685067AF25
    C00.RUJ;1"
    Read/write transaction in progress  ❶
    Transaction sequence number is 136  ❷

Active user with process ID 71A0126C
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 86
    Snapshot transaction in progress
    Transaction sequence number is 136  ❸

Active user with process ID 71A01269
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 166
    Recovery journal filename is "SQL_USER1:[RDM$RUJ]MF_PERSONNEL$0096850694F0
    B7A0.RUJ;1"
    Read/write transaction in progress
    Transaction sequence number is 137  ❹
```

(continued on next page)

Example 4–29 (Cont.) Displaying Transactions When Prestarted Transactions Are Enabled

```
Active user with process ID 71A0126D
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 246
    Snapshot transaction in progress
    Transaction sequence number is 136  ❸
$!
   .
   .
   .
$! After the first read/write transaction (the cutoff TSN) has
$! committed or rolled back:
$ RMU/DUMP/USERS mf_personnel
Active user with process ID 71A01142
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 19
    Recovery journal filename is "SQL_USER1:[RDM$RUJ]MF_PERSONNEL$009685067AF2
    5C00.RUJ;1"
    Read/write transaction in progress  ❺
    Transaction sequence number is 136  ❻

Active user with process ID 71A0126C
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 86
    Snapshot transaction in progress
    Transaction sequence number is 136  ❼

Active user with process ID 71A01269
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 166
    Recovery journal filename is "SQL_USER1:[RDM$RUJ]MF_PERSONNEL$0096850694F0
    B7A0.RUJ;1"
    Read/write transaction in progress
    Transaction sequence number is 137

Active user with process ID 71A0126D
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 246
    Snapshot transaction in progress
    Transaction sequence number is 136  ❼
$
```

The following callout numbers are keyed to Example 4–29.

❶ The first read/write transaction is the oldest read/write transaction in the database, therefore it is the cutoff TSN.

❷ The cutoff TSN is TSN 136.

❸ Read-only (snapshot) transactions receive the TSN of the oldest read/write transaction (the cutoff TSN).

❹ The second read/write transaction receives the next TSN, which is TSN 137.

❺ After the cutoff TSN is committed or rolled back, Oracle Rdb automatically prestarts a new read/write transaction.

❻ The prestarted transaction retains the same TSN as the original transaction, therefore this transaction is still the cutoff TSN.

❼ The oldest read/write transaction in the database is still TSN 136, so Oracle Rdb cannot reclaim the snapshot records written by the cutoff TSN's process and by the other read/write transaction (TSN 137).

Example 4–30 shows the effect that disabling prestarted transactions has for the scenario shown in Figure 4–11. Example 4–30 shows the two read/write transactions and two read-only transactions in Figure 4–11. However, unlike Example 4–29, in this example prestarted transactions are *disabled*. Note that after the first read/write transaction (TSN 136, the cutoff TSN) is committed or rolled back, Oracle Rdb does not begin a prestarted transaction, as shown by the "No transaction in progress" message in the RMU Dump Users output. This allows Oracle Rdb to reclaim snapshot records written before the cutoff TSN committed or rolled back (all snapshot records with TSNs of 136 or less).

Example 4–30 Displaying Transactions When Prestarted Transactions Are Disabled

```
$! While the four transactions are in progress:
$ RMU/DUMP/USERS mf_personnel
Active user with process ID 71A01142
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 19
```

(continued on next page)

Example 4–30 (Cont.) Displaying Transactions When Prestarted Transactions Are Disabled

```
    Recovery journal filename is "SQL_USER1:[RDM$RUJ]MF_PERSONNEL$009685067AF2
    5C00.RUJ;1"
    Read/write transaction in progress  ❶
    Transaction sequence number is 136  ❷

Active user with process ID 71A0126C
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 86
    Snapshot transaction in progress
    Transaction sequence number is 136  ❸

Active user with process ID 71A01269
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 166
    Recovery journal filename is "SQL_USER1:[RDM$RUJ]MF_PERSONNEL$0096850694F0
    B7A0.RUJ;1"
    Read/write transaction in progress
    Transaction sequence number is 137  ❹

Active user with process ID 71A0126D
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 246
    Snapshot transaction in progress
    Transaction sequence number is 136  ❸
$!
   .
   .
   .
$! After the first read/write transaction (the cutoff TSN) has
$! committed or rolled back:
$ RMU/DUMP/USERS mf_personnel
Active user with process ID 71A01142
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 19
    Recovery journal filename is "SQL_USER1:[RDM$RUJ]MF_PERSONNEL$009685067AF2
    5C00.RUJ;1"
    No transaction in progress          ❺

Active user with process ID 71A0126C
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 86
    Snapshot transaction in progress
    Transaction sequence number is 136  ❼
```

(continued on next page)

Example 4–30 (Cont.) Displaying Transactions When Prestarted Transactions Are Disabled

```
Active user with process ID 71A01269
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 166
    Recovery journal filename is "SQL_USER1:[RDM$RUJ]MF_PERSONNEL$0096850694F0
    B7A0.RUJ;1"
    Read/write transaction in progress
    Transaction sequence number is 137  ❻

Active user with process ID 71A0126D
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 246
    Snapshot transaction in progress
    Transaction sequence number is 136  ❼
$
```

The following callout numbers are keyed to Example 4–30.

❶ The first read/write transaction is the oldest read/write transaction in the database, therefore it is the cutoff TSN.

❷ The cutoff TSN is TSN 136.

❸ Read-only (snapshot) transactions receive the TSN of the oldest read/write transaction (the cutoff TSN).

❹ The second read/write transaction receives the next TSN, which is TSN 137.

❺ After the cutoff TSN is committed or rolled back, Oracle Rdb does not prestart a new transaction, as shown by “No transaction in progress” in the display.

❻ The first read/write transaction ends when it is committed or rolled back, and the second read/write transaction (TSN 137) becomes the oldest read/write transaction in the database (the cutoff TSN).

❼ Because the oldest read/write transaction is now TSN 137, Oracle Rdb can reclaim the snapshot records with a TSN of 136 or less. In this case, Oracle Rdb would be able to reclaim the snapshot records by the process with TSN 136.

To enable prestarted transactions, specify or omit the PRESTARTED TRANSACTIONS ARE ON clause of the SQL ATTACH, CONNECT, DECLARE ALIAS, CREATE DATABASE, and IMPORT statements.

To disable prestarted transactions, specify the PRESTARTED TRANSACTIONS ARE OFF clause of the SQL ATTACH, CONNECT, DECLARE ALIAS, CREATE DATABASE, and IMPORT statements.

If the PRESTARTED TRANSACTIONS ARE ON or PRESTARTED TRANSACTIONS ARE OFF clauses are used with the CREATE DATABASE or IMPORT statement, they apply only to the current attach. They do not become permanent database attributes.

See the descriptions of the SQL ATTACH, CONNECT, DECLARE ALIAS, CREATE DATABASE, and IMPORT statements in the *Oracle Rdb7 SQL Reference Manual* for more information on specifying the PRESTARTED TRANSACTIONS ARE ON and PRESTARTED TRANSACTIONS ARE OFF clauses.

4.3 Adjusting Storage Map Parameters

The following sections describe the RMU Analyze Placement command and two storage map parameters (PLACEMENT VIA INDEX and data compression), and provide information you can use to determine optimum values for your database applications. Because you can often attain good performance with Oracle Rdb using the default values for each of these two storage map parameters, you should use these sections as guidelines to adjust your database for applications with unusual characteristics.

Only two storage map parameters have default settings: compression is enabled and logical area thresholds are disabled. All other storage map parameters must be explicitly declared (for example, you must enter a map name).

Table 4–11 describes specific SQL statements and whether or not each statement allows you to specify storage map parameters, using the SQL CREATE DATABASE, ALTER DATABASE, or IMPORT statement, respectively.

Table 4–11 SQL Statements Affecting Storage Map Parameters

Storage Map Parameters	SQL CREATE STORAGE MAP	SQL ALTER STORAGE MAP	SQL IMPORT or CREATE STORAGE MAP
Map name	Yes	Yes	Yes

(continued on next page)

Table 4–11 (Cont.) SQL Statements Affecting Storage Map Parameters

Storage Map Parameters	SQL CREATE STORAGE MAP	SQL ALTER STORAGE MAP	SQL IMPORT or CREATE STORAGE MAP
Description	No	No	No
Specify table	Yes	No	Yes
Specify using column	Yes	Yes	Yes
Specify within storage area name	Yes	Yes	Yes
Specify partitioning limits	Yes	Yes	Yes
Specify PLACEMENT VIA INDEX	Yes	Yes	Yes
Specify compression enable/disable	Yes	Yes	Yes
Specify logical are thresholds	Yes	Yes	Yes
Specify REORGANIZE	No	Yes	No

You can also establish page thresholds for logical areas within a uniform page format using the CREATE or ALTER INDEX and CREATE or ALTER STORAGE MAP statements. Refer to Section 4.2.5 for information on selecting threshold values.

See the *Oracle Rdb7 Guide to Database Design and Definition* for information on modifying many of the storage map parameters and reorganizing data into different storage areas.

4.3.1 Gathering Storage Map Parameter Information

This section describes the format and content of the output when you use the RMU Analyze command and specify the Placement and the Option [= Normal, Full, or Debug] qualifiers. For general information on the RMU Analyze command, see Section 2.1.

The RMU Analyze Placement command is useful to determine the following:

- The maximum and average path length to a data row or stated in another way, the maximum and average number of index records accessed to reach a data row

- The maximum I/O path length or the total number of pages traversed to reach a data row. The maximum I/O path length is the expected I/O when all of the following conditions are true:
  - None of the database records required are buffered (local or global buffers)
  - The minimum number of global buffers are available
  - There is maximum contention from other users for these database records
- The minimum I/O path length or considering the buffer size, whether or not the index and data rows would both be in the buffer. The minimum I/O path length is the expected I/O when all of the following conditions are true:
  - None of the database records required are buffered (local or global buffers)
  - There is an adequate number of buffers to minimize I/O
  - There is no contention from other users for the database records
- The frequency distributions for the dbkey path lengths, maximum I/O path lengths, and minimum I/O path lengths for specified indexes
- The distribution of data rows on data pages in a storage area by logical area ID and dbkey, the number of keys needed to reach each data row, the maximum and minimum I/O path lengths needed to reach the data row, the length of each key, and the specific key for the data row

The estimated maximum I/O path length values are calculated using a worst case scenario, so the real application performance should never be worse than the estimated maximum I/O path length values.

However, in some cases, application performance can be better than the estimated minimum I/O path length values. For example, the second or subsequent index lookup may find all the required records already fetched into buffers by the first index lookup, and little or no I/O may be required. The true minimum I/O path length may be very nearly zero. In most cases, the estimated minimum I/O path length will approximate the true measured I/O path length for the application.

4.3.1.1 Using the RMU Analyze Placement Option=Normal Command

When you use the RMU Analyze Placement command with the Option=Normal qualifier and specify the EMPLOYEES_HASH hashed index, Oracle RMU displays the information shown in Example 4–31.

Example 4–31 RMU Analyze Placement Command with the Option=Normal Qualifier for a Hashed Index

```
$ RMU/ANALYZE/PLACEMENT mf_personnel EMPLOYEES_HASH /OPTION=NORMAL
----------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

----------------------------------------------------------------------------
 Hashed index EMPLOYEES_HASH for relation EMPLOYEES duplicates not allowed
 Levels: 1, Nodes: 69, Keys: 100, Records: 100
 Maximum path length --  DBkeys: 3, IO range: 1 to 1
 Average path length --  DBkeys: 3.00, IO range: 1.00 to 1.00

----------------------------------------------------------------------------
```

The following list explains the information in the output from the RMU Analyze Placement command with the Option=Normal qualifier. Entries in parentheses that follow each field description are keyed to the EMPLOYEES_HASH index shown in Example 4–31.

- Index name (EMPLOYEES_HASH).
- for relation

  Table name (EMPLOYEES).
- duplicates

  Duplicates allowed or not allowed (duplicates not allowed).
- Levels

  The number of levels in the index (1).
- Nodes

  The total number of nodes in the index (69).
- Keys

  The total number of unique keys in the index (100).
- Records

  The total number of data rows with unique keys in the index (100).
- Maximum path length—DBkeys

The maximum number of dbkeys (index records) accessed to reach a data row (3).

- IO range

  The range of the maximum I/O path lengths required to access data rows. The first value represents the low estimate of I/Os required to access data rows. The second value represents the high estimate of I/Os required to access data rows (1 to 1).

- Average path length—DBkeys

  The average number of dbkeys (index records) accessed to reach a data row (3.00).

- IO range

  The range of the average I/O path lengths required to access data rows. The first value represents the low estimate of the average number of I/Os required to access data rows. The second value represents the high estimate of the average number of I/Os required to access data rows (1.00 to 1.00).

When you use the RMU Analyze Placement command with the Option=Normal qualifier and specify the DEPARTMENTS_INDEX sorted index, Oracle RMU displays the information shown in Example 4–32.

Example 4–32 RMU Analyze Placement Command with the Option=Normal Qualifier for a Non-Ranked Sorted Index

```
$ RMU/ANALYZE/PLACEMENT mf_personnel DEPARTMENTS_INDEX /OPTION=NORMAL
----------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

----------------------------------------------------------------------------
 Index DEPARTMENTS_INDEX for relation DEPARTMENTS duplicates not allowed
 Levels: 1, Nodes: 1, Keys: 26, Records: 26
 Maximum path length --  DBkeys: 2, IO range: 1 to 2
 Average path length --  DBkeys: 2.00, IO range: 1.00 to 1.65

----------------------------------------------------------------------------
```

In Example 4–32, for the DEPARTMENTS_INDEX sorted index, the maximum level is 1, and there is one node (the root node). There are 26 unique keys for 26 data rows. The maximum path length (number of records accessed—leaf node and data row) is 2, and the estimated number of I/Os required to access the data with the maximum path length is in the range 1 (low estimate) to 2 (high estimate). The path length (number of records accessed) averaged over all of the indexed data is 2.00, and the estimated number of I/Os required to access the data averaged over all the indexed data is in the range 1.00 (low estimate) to 1.65 (high estimate).

Example 4–33 shows the output from the RMU Analyze Placement command with the Option=Normal when you specify a ranked sorted index that allows duplicates.

Example 4–33 RMU Analyze Placement Command with the Option=Normal Qualifier for a Ranked Sorted Index

```
$ RMU/ANALYZE/PLACEMENT mf_personnel DEGREES_YEAR_RANKED /OPTION=NORMAL
-----------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

-----------------------------------------------------------------------------
 Index DEGREES_YEAR_RANKED for relation DEGREES duplicates allowed
 Levels: 2, Nodes: 5, Keys: 25, Records: 165
   Dup nodes: 0, Dup keys: 18, Dup maps: 18, Dup records: 162
 Maximum path length --  DBkeys: 3, IO range: 2 to 3
 Average path length --  DBkeys: 3.00, IO range: 2.01 to 2.48

-----------------------------------------------------------------------------
```

When an index allows duplicates, the RMU Analyze Placement command displays the following additional information:

- Dup nodes

  For ranked sorted indexes, the number of overflow nodes. The number can be zero (0) even if the index contains duplicates. For non-ranked sorted indexes, the number of duplicate nodes. For hashed indexes, the number of duplicate nodes.

- Dup keys

  The total number of duplicate keys in the index.

- Dup maps

  For ranked sorted indexes only, the number of bit maps used to represent the dbkeys that point to duplicate index key data. This field does not appear for non-ranked sorted indexes or hashed indexes.

- Dup records

  The total number of duplicate records in the index.

When you use the RMU Analyze Placement command with the Option=Normal qualifier and specify the JOB_HISTORY_HASH hashed index, which allows duplicates, Oracle RMU displays the information shown in Example 4–34.

Example 4–34 RMU Analyze Placement Command with the Option=Normal Qualifier for a Hashed Index (Duplicates Allowed)

```
$ RMU/ANALYZE/PLACEMENT mf_personnel JOB_HISTORY_HASH /OPTION=NORMAL
------------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

------------------------------------------------------------------------------
 Hashed index JOB_HISTORY_HASH for relation JOB_HISTORY duplicates allowed
 Levels: 1, Nodes: 69, Keys: 100, Records: 274
   Dup nodes: 80, Dup keys: 80, Dup records: 254
 Maximum path length --  DBkeys: 4, IO range: 1 to 3
 Average path length --  DBkeys: 3.93, IO range: 1.00 to 1.08

------------------------------------------------------------------------------
```

In Example 4–34 for the JOB_HISTORY_HASH hashed index, the level is 1 and there are 69 nodes (hash buckets). There are a total of 100 keys for 274 data rows. There are 80 keys (duplicate node records) that represent 254 data rows. (Thus, there are 20 keys that have no duplicates and represent 20 data records.)

The maximum path length is 4 (number of records accessed—system record, hash bucket, duplicate node record, and data row) and the estimated number of I/Os required to access the data with the maximum path length is in the range 1 (low estimate) to 3 (high estimate). The path length (number of records accessed) averaged over all the indexed data is 3.93, and the estimated number of I/Os required to access the data averaged over all the indexed data is in the range 1.00 (low estimate) to 1.08 (high estimate).

For sorted indexes, if Oracle Rdb reads the top level (root node), the bottom level (leaf node), and the data page, three I/O operations are necessary to make a row retrieval. However, the top level of the index normally stays in the buffer.

If there are enough buffers, the lower levels remain in the buffer as well. For the sorted index DEPARTMENTS_INDEX (Example 4–32), the maximum and average I/O path length ranges are 1 to 2 and 1 to 1.65, respectively. These values indicate that up to 2 pages would be accessed to read any data row. Using the DEPARTMENTS_INDEX, it is almost certain that the top level node record, bottom level node record, and the data row would all be retrieved in the same I/O operation, because both the sorted index structure and all data rows reside on two adjacent pages.

To verify that these records, including the lower level of the sorted index, are all in the buffer, use the RMU Analyze Placement command, specify the Option=Full qualifier, and inspect the MIN I/O path length by frequency histogram. The frequency distribution of minimum I/O path lengths in this histogram generally shows you how accessible data rows are in terms of the number of I/O operations required to access them. Note that when the minimum I/O path length is greater than 1, the top level, the bottom level, and data rows would not be read in one I/O operation.

For hashed indexes stored in the same storage area and on the same page as the data using the PLACEMENT VIA INDEX clause of the SQL CREATE and ALTER STORAGE MAP statement, a minimum of one I/O operation is required to retrieve a data row.

If overflow occurs, data rows may be written to the page if space is available, but if there is not enough space to make entries in the hash bucket on the same page as the data rows, the hash bucket overflows to an adjacent page. Additional data rows that try to hash to the full data page may be written to this adjacent page, and additional entries are made in the overflow hash bucket if space is available for both record types. As a consequence, hash buckets and data may be placed on different pages, which increases the maximum I/O path length by the number of pages on which the hash bucket and data rows have been written.

When duplicates are allowed, duplicate node records are created and also placed on the data page as part of the hash structure. If there are many duplicate records and the page size is too small, data rows are placed on adjacent pages where space is available, followed by the duplicate node records, and finally by more entries in the overflow hash buckets for these data pages. As the database pages fill up over time and more overflows occur, performance may drop as the maximum I/O path length increases to the point where the minimum I/O path length indicates that entire hashed index structures and data rows are no longer in the same buffer and subsequent additional I/O operations are required to gather this information.

The JOB_HISTORY_HASH hashed index (Example 4–34) contains duplicate records and the maximum I/O path length ranges between 1 and 3, but averages 1.00 to 1.08. This indicates that only 1 in 12 data rows have a maximum I/O path length greater than 1 page. In this case, the hash structures and the data rows are all still retrieved in the same I/O operation.

To verify that these duplicate records are all in the buffer together, use the RMU Analyze Placement command, specify the Option=Full qualifier, and inspect the MIN I/O path length by frequency histogram. The frequency distribution of minimum I/O path lengths in this histogram generally shows you how accessible data rows are in terms of the number of I/O operations required to access them. Note that when the minimum I/O path length is greater than 1, all hash structures (system record, hash bucket, overflow hash bucket, and duplicate node records) and data rows are not all read in one I/O operation.

4.3.1.2 Using the RMU Analyze Placement Option=Full Command

When you use the RMU Analyze Placement command and specify the Option=Full qualifier, Oracle RMU displays the following information:

- Summary information

  The summary information is identical to that which results when you specify an index and the Option=Normal qualifier.

- DBkey path length by frequency histogram

  The DBkey path length by frequency histogram shows frequency distribution of the number of dbkeys required to access a data row.

- MAX I/O path length by frequency histogram

  The MAX I/O path length by frequency histogram shows frequency distribution of the maximum number of pages required to reach a data row.

- MIN I/O path length by frequency histogram

  The MIN I/O path length by frequency histogram shows frequency distribution, considering the buffer size, of the instances in which both index and data rows would or would not be in the buffer at the same time.

For the EMPLOYEES_HASH hashed index, the index is only one level. The information for the EMPLOYEES_HASH hashed index is shown in Example 4–35.

Example 4–35 RMU Analyze Placement Option=Full Command for a Hashed Index

```
$ RMU/ANALYZE/PLACEMENT mf_personnel EMPLOYEES_HASH /OPTION=FULL
----------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
```

(continued on next page)

Example 4–35 (Cont.) RMU Analyze Placement Option=Full Command for a Hashed Index

```
----------------------------------------------------------------------------
 Hashed index EMPLOYEES_HASH for relation EMPLOYEES duplicates not allowed
 Levels: 1, Nodes: 69, Keys: 100, Records: 100
 Maximum path length --  DBkeys: 3, IO range: 1 to 1
 Average path length --  DBkeys: 3.00, IO range: 1.00 to 1.00

                        DBkey path length vs. frequency

     6 | (0)
     5 | (0)
     4 | (0)
     3 |====================================================== (100)
     2 | (0)
     1 | (0)

                       MAX IO path length vs. frequency

     6 | (0)
     5 | (0)
     4 | (0)
     3 | (0)
     2 | (0)
     1 |====================================================== (100)

                       MIN IO path length vs. frequency

     6 | (0)
     5 | (0)
     4 | (0)
     3 | (0)
     2 | (0)
     1 |====================================================== (100)

----------------------------------------------------------------------------
```

For the DEPARTMENTS_INDEX sorted index, the index is only one level. The information for the DEPARTMENTS_INDEX sorted index is shown in Example 4–36.

Example 4–36 RMU Analyze Placement Option=Full Command for a Sorted Index

```
$ RMU/ANALYZE/PLACEMENT mf_personnel DEPARTMENTS_INDEX /OPTION=FULL
-----------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

-----------------------------------------------------------------------------
 Index DEPARTMENTS_INDEX for relation DEPARTMENTS duplicates not allowed
 Levels: 1, Nodes: 1, Keys: 26, Records: 26
 Maximum path length --  DBkeys: 2, IO range: 1 to 2
 Average path length --  DBkeys: 2.00, IO range: 1.00 to 1.65

                        DBkey path length vs. frequency

     6 | (0)
     5 | (0)
     4 | (0)
     3 | (0)
     2 |======================================================= (26)
     1 | (0)


                       MAX IO path length vs. frequency

     6 | (0)
     5 | (0)
     4 | (0)
     3 | (0)
     2 |======================================================= (17)
     1 |============================ (9)


                       MIN IO path length vs. frequency

     6 | (0)
     5 | (0)
     4 | (0)
     3 | (0)
     2 | (0)
     1 |======================================================= (26)

-----------------------------------------------------------------------------
```

The information for the JOB_HISTORY_HASH hashed index is shown in Example 4–37.

Example 4–37 RMU Analyze Placement Option=Full Command for a Hashed Index (Duplicates Allowed)

```
$ RMU/ANALYZE/PLACEMENT mf_personnel JOB_HISTORY_HASH /OPTION=FULL
----------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

----------------------------------------------------------------------------
 Hashed index JOB_HISTORY_HASH for relation JOB_HISTORY duplicates allowed
 Levels: 1, Nodes: 69, Keys: 100, Records: 274
   Dup nodes: 80, Dup keys: 80, Dup records: 254
 Maximum path length --  DBkeys: 4, IO range: 1 to 3
 Average path length --  DBkeys: 3.93, IO range: 1.00 to 1.08

                        DBkey path length vs. frequency

     6 | (0)
     5 | (0)
     4 |====================================================== (254)
     3 |==== (20)
     2 | (0)
     1 | (0)


                       MAX IO path length vs. frequency

     6 | (0)
     5 | (0)
     4 | (0)
     3 | (1)
     2 |==== (21)
     1 |====================================================== (252)


                       MIN IO path length vs. frequency

     6 | (0)
     5 | (0)
     4 | (0)
     3 | (0)
     2 | (0)
     1 |====================================================== (274)

----------------------------------------------------------------------------
```

Further analysis of the three indexes, EMPLOYEES_HASH, DEPARTMENTS_INDEX, and JOB_HISTORY_HASH using the three histograms yields the following information:

- The EMPLOYEES_HASH hashed index (Example 4–35)

  All 100 data rows have a dbkey path length of 3 dbkeys (system record, hash bucket, and data row). The maximum I/O path length (number of pages accessed) for all 100 index records is 1. This means that each respective EMPLOYEES_HASH hash bucket shares the same page as its EMPLOYEES data row. The minimum I/O path length for all 100 index records is 1. This means, considering the buffer size, that for any EMPLOYEES_HASH hash bucket, the respective EMPLOYEES data row would be retrieved in one I/O operation.

- The DEPARTMENTS_INDEX sorted index (Example 4–36)

  All 26 data rows have a dbkey path length of 2 dbkeys (leaf node and data row). The maximum I/O path length (number of pages accessed) for all 26 index records is 2 for 17 index records and 1 for 9 index records. This means that 9 DEPARTMENTS_INDEX leaf node records share the same page as their respective DEPARTMENTS data row and that 17 DEPARTMENTS data rows can be accessed by traversing 2 pages. The minimum I/O path length for all 26 index records is 1. This means, considering the buffer size, that for any DEPARTMENTS_INDEX leaf node record, the respective DEPARTMENTS data row would be retrieved in one I/O operation.

- The JOB_HISTORY_HASH hashed index (Example 4–37)

  For 274 data rows, the dbkey path length is as follows: 3 dbkeys for 20 index records (system record, hash bucket, and data row) and 4 dbkeys for 254 records (system record, hash bucket, duplicate node record, and data row). The maximum I/O path length (number of pages accessed) for 274 index records is 1 for 252 index records, 2 for 21 index records, and 3 for 1 index record. This means that most JOB_HISTORY_HASH hash buckets (252) share the same page as their respective JOB_HISTORY data row, while 21 rows can be accessed by traversing 2 pages, and 1 data row can be accessed by traversing 3 pages. The minimum I/O path length for all 274 index records is 1. This means, considering the buffer size, that for any JOB_HISTORY hash bucket, the respective JOB_HISTORY data row would be retrieved in one I/O operation.

  Note that because duplicates are allowed for the JOB_HISTORY_HASH hashed index, 20 data rows with no duplicates can be reached by accessing 3 dbkeys, and 254 data rows can be reached by accessing 4 dbkeys that include the duplicate node records.

Because pages filled with hashed index structures and their respective associated parent-child data rows, the maximum I/O path length indicated that a small percentage of hash buckets (8 percent or 22 index records) were not on the same page as their respective JOB_HISTORY data rows. By following the DBkey path length by frequency histogram and the MAX I/O path length by frequency histogram over time, you can observe how the dbkey path length frequency might increase, especially for values of 5 dbkeys or higher, and how the maximum I/O path length frequency might also increase for values of 2 or higher. However, more importantly, you should follow the MIN I/O path length by frequency histogram and determine if hash structures and data rows would still be retrieved in one I/O operation. When the minimum I/O path length values become greater than 1, you may notice some performance degradation as more I/O operations are necessary to retrieve both index and data rows.

4.3.1.3 Using the RMU Analyze Placement Option=Debug Command

When you use the RMU Analyze Placement command with the Option=Debug qualifier, Oracle RMU displays detailed index node and index record information:

- Distribution of data rows on data pages
- Number of keys needed to reach a specific data row on a data page
- Maximum I/O path length to access a data row (number of pages traversed)
- Minimum I/O path length to access a row when considering the buffer size
- Length of each key, and the specific key for the data row

The output header is divided into two parts, general and detailed information. The headers are followed by detailed index record information. The last part of the output, including the three histograms, displays information identical to the RMU Analyze Placement command with the Option=Full qualifier. If an index is stored in more than one logical area, information for all logical areas is displayed in the output.

Example 4–38 shows the output when you use the RMU Analyze Placement command with the Option=Debug qualifier and specify the EMPLOYEES_HASH hashed index.

Example 4–38 RMU Analyze Placement Option=Debug Command for a Hashed Index

```
$ RMU/ANALYZE/PLACEMENT mf_personnel EMPLOYEES_HASH /OPTION=DEBUG
0----------------------------------------------------------------------------
0
0 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
0
0   ID   Hash Dup Name                           Relation
0----------------------------------------------------------------------------
4      62   T   F  EMPLOYEES_HASH                 EMPLOYEES
4      60   T   F  EMPLOYEES_HASH                 EMPLOYEES
4      58   T   F  EMPLOYEES_HASH                 EMPLOYEES
0
0----------------------------------------------------------------------------
0
0 Selected areas -
0 Area =   1 - RDB$SYSTEM
0 Area =   2 - EMPIDS_LOW
0 Area =   3 - EMPIDS_MID
0 Area =   4 - EMPIDS_OVER
0 Area =   5 - DEPARTMENTS
0 Area =   6 - SALARY_HISTORY
0 Area =   7 - JOBS
0 Area =   8 - EMP_INFO
0 Area =   9 - RESUME_LISTS
0 Area =  10 - RESUMES
0
0----------------------------------------------------------------------------
0
0   ID        DB KEY      DB_KEYS  MAX_IO  MIN_IO (LEN)"KEY"
0----------------------------------------------------------------------------
0
7  58 0063:0000000002:001       3      1       1 (  6)"003030313635"
7  58 0063:0000000002:003       3      1       1 (  6)"003030313930"
7  58 0063:0000000005:001       3      1       1 (  6)"003030313837"
7  58 0063:0000000007:001       3      1       1 (  6)"003030313639"
7  58 0063:0000000007:003       3      1       1 (  6)"003030313736"
7  58 0063:0000000007:004       3      1       1 (  6)"003030313938"
   .
   .
   .
```

(continued on next page)

Example 4–38 (Cont.) RMU Analyze Placement Option=Debug Command for a Hashed Index

```
7  60 0064:0000000003:001       3      1       1 (  6)"003030323133"
7  60 0064:0000000004:001       3      1       1 (  6)"003030323139"
7  60 0064:0000000005:001       3      1       1 (  6)"003030323235"
7  60 0064:0000000005:003       3      1       1 (  6)"003030323430"
7  60 0064:0000000006:001       3      1       1 (  6)"003030323637"
7  60 0064:0000000009:001       3      1       1 (  6)"003030323837"
   .
   .
   .
7  62 0065:0000000008:001       3      1       1 (  6)"003030343135"
7  62 0065:0000000018:001       3      1       1 (  6)"003030343335"
7  62 0065:0000000021:001       3      1       1 (  6)"003030343035"
7  62 0065:0000000026:001       3      1       1 (  6)"003030343138"
7  62 0065:0000000032:001       3      1       1 (  6)"003030343731"
7  62 0065:0000000046:001       3      1       1 (  6)"003030343136"
----------------------------------------------------------------------------
 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

----------------------------------------------------------------------------
0 Hashed index EMPLOYEES_HASH for relation EMPLOYEES duplicates not allowed
0 Levels: 1, Nodes: 69, Keys: 100, Records: 100
0 Maximum path length --  DBkeys: 3, IO range: 1 to 1
0 Average path length --  DBkeys: 3.00, IO range: 1.00 to 1.00
   .
   .
   .
```

The following list provides information about the fields in the RMU Analyze Placement command with the Option=Debug qualifier. Entries in parentheses that follow each field description are keyed to the EMPLOYEES_HASH index, logical area 58, shown in Example 4–38.

- General index information
  - ID

    The logical area ID for the index (58).
  - Hash

    A coded field: T = TRUE for hash index, F = FALSE for B-tree index (T).
  - Dup

    A coded field: T = TRUE for duplicates allowed, F = FALSE for duplicates not allowed (F).

  - Name

    Name of the index (EMPLOYEES_HASH).

  - Relation

    The name of the table for which the index is to be used (EMPLOYEES).

- The selected areas legend

  A legend that denotes the storage areas that comprise the mf_personnel database.

- Detailed index information

  - ID

    The logical area ID for the index (58).

  - DB KEY

    The dbkey for the record; it is comprised of three parts—the logical area ID (0063), the page number (0000000002), and the line on the page where the record is stored (001).

  - DB_KEYS

    The number of keys needed to access the data row (3); system record, hash bucket, and data row dbkeys.

  - MAX_IO

    The maximum I/O path length or the total number of pages traversed to access a data row (1).

  - MIN_IO

    The minimum I/O path length. The value 1 indicates that the index and data row could both be in the buffer (1).

  - (LEN)

    The length of the key in bytes (6).

  - “KEY”

    The actual key printed as a hexadecimal string (003030313635).

When you use the RMU Analyze Placement command with the Option=Debug qualifier and specify the DEPARTMENTS_INDEX sorted index, Oracle RMU displays the information shown in Example 4–39.

Example 4–39 RMU Analyze Placement Option=Debug Command for a Sorted Index

```
$ RMU/ANALYZE/PLACEMENT mf_personnel DEPARTMENTS_INDEX /OPTION=DEBUG
0----------------------------------------------------------------------------
0
0 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
0
0   ID   Hash Dup Name                           Relation
0----------------------------------------------------------------------------
4      72   F   F  DEPARTMENTS_INDEX              DEPARTMENTS
0
0----------------------------------------------------------------------------
0
0 Selected areas -
0 Area =   1 - RDB$SYSTEM
0 Area =   2 - EMPIDS_LOW
0 Area =   3 - EMPIDS_MID
0 Area =   4 - EMPIDS_OVER
0 Area =   5 - DEPARTMENTS
0 Area =   6 - SALARY_HISTORY
0 Area =   7 - JOBS
0 Area =   8 - EMP_INFO
0 Area =   9 - RESUME_LISTS
0 Area =  10 - RESUMES
0
0----------------------------------------------------------------------------
0
0   ID        DB KEY      DB_KEYS  MAX_IO  MIN_IO (LEN)"KEY"
0----------------------------------------------------------------------------
0
7  72 0073:0000000002:001       2      1       1 (  5)"0041444D4E"
7  72 0073:0000000002:003       2      1       1 (  5)"00454C454C"
7  72 0073:0000000002:004       2      1       1 (  5)"00454C4753"
7  72 0073:0000000002:005       2      1       1 (  5)"00454C4D43"
7  72 0073:0000000002:006       2      1       1 (  5)"00454E4720"
7  72 0073:0000000002:007       2      1       1 (  5)"004D424D46"
7  72 0073:0000000002:008       2      1       1 (  5)"004D424D4E"
7  72 0073:0000000002:009       2      1       1 (  5)"004D424D53"
7  72 0073:0000000003:001       2      2       1 (  5)"004D43424D"
```

(continued on next page)

Example 4–39 (Cont.) RMU Analyze Placement Option=Debug Command for a Sorted Index

```
7  72 0073:0000000003:002       2      2       1 (  5)"004D434253"
7  72 0073:0000000003:003       2      2       1 (  5)"004D475654"
7  72 0073:0000000003:004       2      2       1 (  5)"004D4B5447"
7  72 0073:0000000003:005       2      2       1 (  5)"004D4E4647"
7  72 0073:0000000003:006       2      2       1 (  5)"004D534349"
7  72 0073:0000000003:007       2      2       1 (  5)"004D534D47"
7  72 0073:0000000003:008       2      2       1 (  5)"004D54454C"
7  72 0073:0000000003:009       2      2       1 (  5)"005045524C"
7  72 0073:0000000003:010       2      2       1 (  5)"0050455253"
7  72 0073:0000000003:011       2      2       1 (  5)"005048524E"
7  72 0073:0000000003:012       2      2       1 (  5)"0050524D47"
7  72 0073:0000000003:013       2      2       1 (  5)"0053414C45"
7  72 0073:0000000003:014       2      2       1 (  5)"0053455552"
7  72 0073:0000000003:015       2      2       1 (  5)"0053554E45"
7  72 0073:0000000003:016       2      2       1 (  5)"0053555341"
7  72 0073:0000000003:017       2      2       1 (  5)"005355534F"
7  72 0073:0000000002:010       2      1       1 (  5)"0053555745"
0-----------------------------------------------------------------------------
0
0 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
0
0-----------------------------------------------------------------------------
0 Index DEPARTMENTS_INDEX for relation DEPARTMENTS duplicates not allowed
0 Levels: 1, Nodes: 1, Keys: 26, Records: 26
0 Maximum path length --  DBkeys: 2, IO range: 1 to 2
0 Average path length --  DBkeys: 2.00, IO range: 1.00 to 1.65
   .
   .
   .
```

In Example 4–39, the DEPARTMENTS_INDEX sorted index is for the DEPARTMENTS table. The entire index (26 records) is located on pages 2 and 3 in logical area 72. The first dbkey, 0073:0000000002:001, indicates that the data row it is pointing to is in logical area 73, on page 2, line entry 1. The total number of dbkeys needed to access this data row is 2 (leaf node and data row). The maximum I/O path length (number of pages accessed) is 1. The minimum I/O path length (considering the buffer) is 1. The key length is 5 bytes, and the key value printed as a hexadecimal string is "0041444D4E".

For sorted indexes, if the index node is small and there are no duplicates, very little system resources are expended to keep the index on the table.

Generally, you should pay close attention to the length of the index node and to the number of duplicates, if they are allowed. If the index node is long, you may be using a lot of system resources to keep the index on the table. At the same time, if you allow duplicates and there are many duplicate values for the index, the efficiency of the index is reduced.

By using the Option=Debug qualifier, you can find specifically where problems are occurring. Problems are initially indicated when you use the Option=Full qualifier. You should check the values for dbkeys, maximum I/O path lengths, and minimum I/O path lengths on a regular basis and note changes and trends.

When you use the RMU Analyze Placement command with the Option=Debug qualifier and specify the JOB_HISTORY_HASH hashed index, Oracle RMU displays the information shown in Example 4–40.

Example 4–40 RMU Analyze Placement Option=Debug Command for a Hashed Index (Duplicates Allowed)

```
$ RMU/ANALYZE/PLACEMENT mf_personnel JOB_HISTORY_HASH /OPTION=DEBUG
0----------------------------------------------------------------------------
0
0 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;
0
0   ID   Hash Dup Name                           Relation
0----------------------------------------------------------------------------
4      68   T   T  JOB_HISTORY_HASH               JOB_HISTORY
4      67   T   T  JOB_HISTORY_HASH               JOB_HISTORY
4      66   T   T  JOB_HISTORY_HASH               JOB_HISTORY
0
0----------------------------------------------------------------------------
0
0 Selected areas -
0 Area =   1 - RDB$SYSTEM
0 Area =   2 - EMPIDS_LOW
0 Area =   3 - EMPIDS_MID
0 Area =   4 - EMPIDS_OVER
0 Area =   5 - DEPARTMENTS
0 Area =   6 - SALARY_HISTORY
0 Area =   7 - JOBS
0 Area =   8 - EMP_INFO
0 Area =   9 - RESUME_LISTS
0 Area =  10 - RESUMES
0
0----------------------------------------------------------------------------
```

(continued on next page)

Example 4–40 (Cont.) RMU Analyze Placement Option=Debug Command for a Hashed Index (Duplicates Allowed)

```
0
0   ID        DB KEY      DB_KEYS  MAX_IO  MIN_IO (LEN)"KEY"
0----------------------------------------------------------------------------
7  66 0069:0000000002:009       4      1       1 (  6)"003030313635"
7  66 0069:0000000002:008       4      1       1 (  6)"003030313635"
7  66 0069:0000000002:006       4      1       1 (  6)"003030313635"
7  66 0069:0000000002:004       4      1       1 (  6)"003030313635"
7  66 0069:0000000002:014       4      1       1 (  6)"003030313930"
7  66 0069:0000000002:013       4      1       1 (  6)"003030313930"
7  66 0069:0000000002:011       4      1       1 (  6)"003030313930"
7  66 0069:0000000002:010       4      1       1 (  6)"003030313930"
7  66 0069:0000000005:006       3      1       1 (  6)"003030313837"
7  66 0069:0000000007:010       4      1       1 (  6)"003030313639"
7  66 0069:0000000007:009       4      1       1 (  6)"003030313639"
7  66 0069:0000000007:007       4      1       1 (  6)"003030313639"
7  66 0069:0000000007:005       4      1       1 (  6)"003030313639"
7  66 0069:0000000005:005       4      2       1 (  6)"003030313736"
   .
   .
   .
7  67 0070:0000000003:003       3      1       1 (  6)"003030323133"
7  67 0070:0000000004:007       4      1       1 (  6)"003030323139"
7  67 0070:0000000004:005       4      1       1 (  6)"003030323139"
7  67 0070:0000000004:003       4      1       1 (  6)"003030323139"
   .
   .
   .
7  68 0071:0000000008:008       4      1       1 (  6)"003030343135"
7  68 0071:0000000008:007       4      1       1 (  6)"003030343135"
7  68 0071:0000000008:005       4      1       1 (  6)"003030343135"
7  68 0071:0000000008:003       4      1       1 (  6)"003030343135"
7  68 0071:0000000018:007       4      1       1 (  6)"003030343335"
----------------------------------------------------------------------------

 Indices for database  - $DUA0:[ORION]MF_PERSONNEL.RDB;

----------------------------------------------------------------------------
```

(continued on next page)

Example 4–40 (Cont.) RMU Analyze Placement Option=Debug Command for a Hashed Index (Duplicates Allowed)

```
0 Hashed index JOB_HISTORY_HASH for relation JOB_HISTORY duplicates allowed
0 Levels: 1, Nodes: 69, Keys: 100, Records: 274
0   Dup nodes: 80, Dup keys: 80, Dup records: 254
0 Maximum path length --  DBkeys: 4, IO range: 1 to 3
0 Average path length --  DBkeys: 3.93, IO range: 1.00 to 1.08
   .
   .
   .
```

In Example 4–40 for the JOB_HISTORY_HASH hashed index, the index is partitioned across three logical areas (66, 67, and 68) for the table JOB_HISTORY. The first dbkey, 0069:0000000002:009, indicates that the data row it is pointing to is in logical area 69, on page 2, line entry 9. The total number of dbkeys needed to access this data row is 4 (system record, hash bucket, duplicate node record, and data row). The maximum I/O path length (number of pages accessed) is 1. The minimum I/O path length (considering the buffer) is 1. The key length is 6 bytes, and the key value printed as a hexadecimal string is "003030313635".

Because duplicates are allowed for the JOB_HISTORY_HASH hashed index, duplicate node records must be accessed to reach a data row and the number of dbkeys increases to 4. On occasion, the maximum I/O path length increases to 2. This indicates that the data row resides on a different page from its hash bucket or duplicate node record.

As pages fill, closely monitor the minimum I/O path length values. If values become greater than 1, more I/O operations will be necessary to access the data row, because both the index record and data row will no longer be in the buffer together. This is an early sign of performance degradation.

For hashed indexes, when hash bucket overflows occur, especially for a database that is growing with many new parent and child duplicate records, detailed placement information should be monitored on a regular basis to determine the nature and extent of any hash bucket overflows.

The first indication that you may have a hash bucket overflow is when the maximum index level changes from a value of 1 to 2 or higher, based on the results of either an RMU Analyze Indexes command or an RMU Analyze Placement command. You should also note all instances when values obtained for the minimum I/O path length for any data row become greater than an expected value of 1. A value of 2 or higher for the minimum I/O path length indicates that two or more I/O operations are necessary to retrieve all associated hash index records and the data row, because none of these records are physically close enough together on the disk to be accessed in one I/O operation. This situation subsequently leads to decreased performance for your database application.

Be careful when estimating the page size and the number of child duplicate records initially to be stored on a page when the PLACEMENT VIA INDEX option is used. To store both parent and child data rows and any defined hash structures on the same page, you must select a page size that will initially provide adequate space to hold these records, as well as sufficient space in which to add more records over some planned period of time.

To produce custom reports to pinpoint specific problems, you could use the output from RMU Analyze Placement command with the Option=Debug qualifier as input to a program of your own design. For example, you could scan the output for instances of where values for dbkeys are greater than some value you select, or where values for the minimum I/O path length were greater than 1, and print the entire record for each instance. Remember, however, that the three histograms produced from the RMU Analyze Placement command with the Option=Full qualifier give you dbkey path length, maximum I/O path length, and minimum I/O path length frequency distribution information. These histograms provide a sufficient overview of your data to give you the first obvious indications of access problems.

4.3.2 PLACEMENT VIA INDEX Option

The PLACEMENT VIA INDEX option permits you to specify the name of an index to be used for placing rows into the table specified in the SQL CREATE STORAGE MAP statement. Rows are stored in storage areas according to the specifications of the STORE clause. Using a placement index to initially store rows in a specific order improves access to these rows. This improved performance continues as long as there is sufficient space on the data page in which to store new rows. Sufficient space on the data page can be ensured for a period of time by carefully calculating the page and file allocation sizes needed. Using mixed storage areas, for example, with a hashed placement index along with a properly sized page and file allocation to contain both the index structures and the data rows, provides the level of tuning required to achieve one I/O operation to retrieve a row in an exact-match query.

If your data pages start to fill up with rows and spill over to adjacent data pages, the performance gains of using an index and the PLACEMENT VIA INDEX option may slowly begin to drop. You can regain the performance edge by using the SQL ALTER DATABASE statement and defining new storage areas with comparable specifications. This causes a reload of the data from these less efficient storage areas into the new storage areas when you modify the storage maps with the SQL ALTER STORAGE MAP statement so that each storage map points to its respective new storage areas. This process is described in more detail in the *Oracle Rdb7 Guide to Database Maintenance*.

4.3.3 Data Compression Option for a Table

Data compression allows Oracle Rdb to fit more data in fewer disk blocks. This saves space and may reduce the time it takes Oracle Rdb to perform certain kinds of retrievals. When you create new tables, use the SQL CREATE STORAGE MAP statement and specify the ENABLE COMPRESSION option to set this characteristic. Use the SQL ALTER STORAGE MAP statement specifying the table name and the DISABLE COMPRESSION option to automatically change the compression characteristic for the table rows in the storage area specified in the STORE clause. Alternatively, use the SQL EXPORT and IMPORT statements to change this characteristic or define a new storage area, specify a new compression parameter, and dynamically remap rows to the new storage area with the SQL ALTER STORAGE MAP statement. Use this alternative, especially if other major database changes are planned and you want to implement them all in one operation.

In general, disable compression only if:

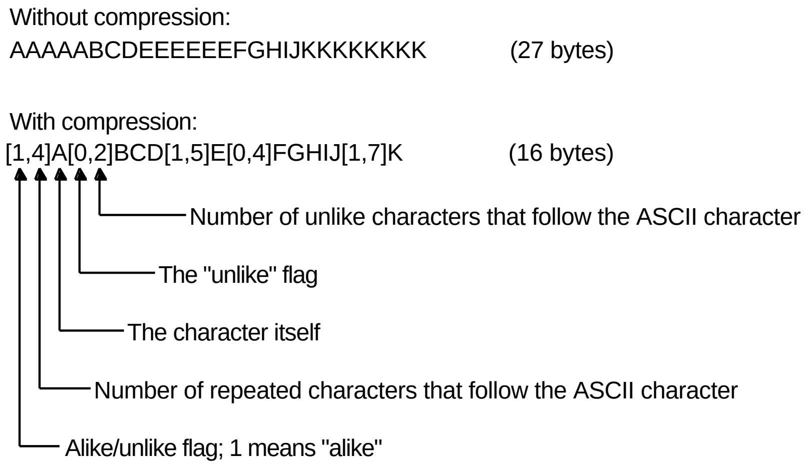
- You are getting lots of row fragmentation
- You have very stable row sizes with no benefit from compression

In both instances, use the RMU Analyze command to check for row fragmentation and row compression benefits in storage areas. See Section 2.1 for more information on how to use the RMU Analyze command and how to interpret the results.

Generally, data compression works well for tables that are not updated frequently. This type of table contains fairly stable information and is accessed primarily for ad hoc queries and sequential retrievals. With compressed data, more data can be placed in fewer disk blocks; this saves disk space, and sequential read operations bring more data into the user's buffer with each I/O operation to a disk.

If a table is frequently updated, it does not usually benefit from data compression. For example, in an order-entry application, a table named NEXT_ORDER_NUMBER has one row used by all users to get the sequence number. Because this row is frequently updated, it is not practical to compress this row. In insert and update operations, data compression costs more CPU time because Oracle Rdb uses the data compression algorithm to format input. Any tables that users modify and query fairly often become fragmented if the row size changes (increases). Retrieving a row that Oracle Rdb has fragmented over one or more database pages may slow the read operation dramatically. If the size of the row has changed significantly, the time it takes to complete a read operation may be even longer. For more information on data compression, see the *Oracle Rdb7 SQL Reference Manual*. Figure 4–12 illustrates data as it would be represented without and with compression. Instead of the ASCII code, the character itself is shown.

Figure 4–12 Effect of Data Compression


Without compression:
AAAAABCDEEEEEEFGHIJKKKKKKKK
(27 bytes)
With compression:
[1,4]A[0,2]BCD[1,5]E[0,4]FGHIJ[1,7]K
(16 bytes)
Number of unlike characters that follow the ASCII character
The "unlike" flag
The character itself
Number of repeated characters that follow the ASCII character
Alike/unlike flag; 1 means "alike"

ZK−7400−GE

Note that the information illustrated here in brackets occupies 1 byte. Obviously, the brackets are not part of the stored data. The designation of [1,4]A means:

- 1

  The following string contains at least three consecutive characters.

- 4

  There are five characters that are alike (the letter A). This designation is zero-based; thus the 4 indicates that five characters are present.

- A

  An ASCII value.

The designation of [0,2]BCD means:

- 0

  The following string does not contain alike characters.

- 2

  There are three unlike characters (the letters BCD); again, the designation is zero-based.

Because not all types of tables benefit from data compression, you can enable and disable compression for each table.

Row fragmentation may occur when a modification extends the row's physical length. Such an extension causes the row to fragment if the database page cannot contain the size of the new storage segment.

Caution

Avoid disabling data compression for tables that contain SQL VARCHAR data type columns. When data compression for an SQL VARCHAR data type column is disabled, Oracle Rdb stores the column's maximum length. Thus, if you defined an SQL VARCHAR data type column as 32,000 bytes, and the column contained 1000 nonblank characters and 31,000 blanks, Oracle Rdb stores 32,000 characters. Oracle Rdb ignores the count portion of an SQL VARCHAR data type column and always treats it as the column's maximum length.

4.3.3.1 Examples of Setting Data Compression

This section shows examples of setting data compression characteristics.

Example 4–41, for single-file databases only, defines a new EMPLOYEES_MAP storage map and disables compression for the EMPLOYEES table.

Example 4–41 Setting Data Compression for Single-File Databases

```
SQL> CREATE STORAGE MAP EMPLOYEES_MAP
cont>   STORE IN ...
cont>   DISABLE COMPRESSION;
SQL>
SQL> SHOW STORAGE MAPS EMPLOYEES_MAP
     EMPLOYEES
 For Table:             EMPLOYEES
 Compression is:        DISABLED
 Store clause:          STORE IN ...
```

In Example 4–42, assume that data compression is disabled for the EMPLOYEES table and enabled (by default) for the SALARY_HISTORY table. The goal is to continue disabled data compression for the EMPLOYEES table and to change the SALARY_HISTORY table from enabled to disabled.

An efficient way to do this is to specify DISABLE COMPRESSION in the SQL ALTER STORAGE MAP statement. You automatically change compression for all rows for the table specified in the original storage map definition statement for the storage area specified in the STORE clause of the SQL ALTER STORAGE MAP statement, as shown in Example 4–42.

Example 4–42 Setting Data Compression for Tables Using a Storage Map Statement

```
SQL> ALTER STORAGE MAP SALARY_HISTORY_MAP
cont>   STORE IN SALARY_HISTORY
cont>   DISABLE COMPRESSION;
```

However, you must be certain that the uncompressed rows still fit on the page; otherwise rows may become fragmented, reducing performance. This highlights the importance of defining page sizes based on uncompressed row sizes. If you know rows will become fragmented when compression is disabled for a storage area, you need to define a new storage area and move all the rows to this new storage area in which the page size is specified correctly. For storage areas with mixed page format, you could just decrease the SPAM threshold values using the THRESHOLDS ARE option so fewer rows are initially stored per page.

To accomplish the combined goal of decompressing rows and avoiding fragmentation due to a small page size, make these changes:

1. Within an SQL ALTER DATABASE statement add-storage-area-clause, create a new storage area using new names for the storage area and storage area file, and either specify a larger page size or a larger SPAM interval.
2. Modify the SALARY_HISTORY_MAP storage map using the SQL ALTER STORAGE MAP statement and change the STORE clause so it points to the new storage area. Set the enable/disable data compression characteristic to disable. When these changes are made, rows from the SALARY_HISTORY table are moved to the new storage area and take on the desired characteristics specified in the ALTER DATABASE, CREATE STORAGE AREA, and ALTER STORAGE MAP statements.
3. Delete the old storage area.

Example 4–43 performs each of these steps.

Example 4–43 Defining a New Storage Area and Specifying a Larger Page Size, Decompressing Rows in a Storage Map, and Deleting the Old Storage Area

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>   ADD STORAGE AREA NEW_SALARY_HISTORY FILENAME new_salary_history.rda
cont>       ALLOCATION IS 25 PAGES
cont>       PAGE SIZE IS 3 BLOCKS
cont>       PAGE FORMAT IS MIXED
cont>       SNAPSHOT FILENAME new_salary_history.snp
cont>       SNAPSHOT ALLOCATION IS 10 PAGES;
SQL>
SQL> ATTACH 'FILENAME mf_personnel';
SQL>
SQL> ALTER STORAGE MAP SALARY_HISTORY_MAP
cont>   STORE IN NEW_SALARY_HISTORY
cont>   DISABLE COMPRESSION;
SQL>
SQL> COMMIT;
SQL> DISCONNECT ALL;
SQL>
SQL> ALTER DATABASE FILENAME mf_personnel
cont>   DROP STORAGE AREA SALARY_HISTORY;
```

You can specify changes such as disabled compression with the SQL EXPORT and IMPORT statements as shown by the SQL procedure in Example 4–44. This method is useful if you have many other changes planned including, perhaps, changing the buffer size. Because the default is ENABLE COMPRESSION, if DISABLE COMPRESSION is not specified in the SQL ALTER STORAGE MAP statement, DISABLE COMPRESSION must be specified in the storage map statements for both the EMPLOYEES and the SALARY_HISTORY tables.

Example 4–44 Using the IMPORT Statement to Specify Import Parameters, New Storage Area, and Storage Map Characteristics

```
SQL> EXPORT DATABASE FILENAME mf_personnel.rdb INTO intpers_bu.rbr;
SQL> --
SQL> -- Export uses WITH EXTENSIONS by default
SQL> --
SQL> @IMPORT_COMPRESS
```

(continued on next page)

Example 4–44 (Cont.) Using the IMPORT Statement to Specify Import Parameters, New Storage Area, and Storage Map Characteristics

```
--
IMPORT DATABASE FROM intpers_bu.rbr
--
-- Specify import options
--
NO ACL
--
-- Specify root-file parameters
FILENAME newtest.rdb
BUFFER SIZE IS 12 BLOCKS
--
-- Specify storage area parameters
-- Define the new storage areas needed
--
CREATE STORAGE AREA EMPIDS_LOW FILENAME new_empids_low.rda
--
-- Specify needed options for the new storage area
--
  ALLOCATION IS 50 PAGES
  PAGE FORMAT IS MIXED
  SNAPSHOT FILENAME new_empids_low.snp
  SNAPSHOT ALLOCATION IS 10 PAGES
--
CREATE STORAGE AREA NEW_SALARY_HISTORY FILENAME new_salary_history.rda
  ALLOCATION IS 25 PAGES
  PAGE SIZE IS 3 BLOCKS
  PAGE FORMAT IS MIXED
  SNAPSHOT FILENAME new_salary_history.snp
  SNAPSHOT ALLOCATION IS 10 PAGES
   .
   .
   .
-- Specify metadata options by specifying new storage map options with
-- the CREATE STORAGE MAP statement for the EMPLOYEES_MAP and the
-- SALARY_HISTORY_MAP
--
CREATE STORAGE MAP EMPLOYEES_MAP FOR EMPLOYEES
  STORE USING EMPLOYEE_ID
    IN
    EMPIDS_LOW WITH LIMIT OF ('00200')
    EMPIDS_MID WITH LIMIT OF ('00400')
    OTHERWISE IN EMPIDS_OVER
```

(continued on next page)

Example 4–44 (Cont.) Using the IMPORT Statement to Specify Import Parameters, New Storage Area, and Storage Map Characteristics

```
    PLACEMENT VIA INDEX EMPLOYEES_HASH
  DISABLE COMPRESSION
--
CREATE STORAGE MAP SALARY_HISTORY_MAP FOR SALARY_HISTORY
  STORE IN NEW_SALARY_HISTORY
  DISABLE COMPRESSION
   .
   .
   .
END IMPORT;
```

Example 4–45 shows the messages displayed by the IMPORT statement when the SQL procedure executes.

Example 4–45 IMPORT Statement Messages

```
Exported by Oracle Rdb V7.0-00 Import/Export utility
A component of SQL V7.0-00
Previous name was mf_personnel.rdb
It was logically exported on 28-MAY-1996 13:25
   .
   .
   .
IMPORTing table EMPLOYEES
IMPORTing table SALARY_HISTORY
   .
   .
   .
-- Import complete
```

Then, use the SQL SHOW STORAGE MAP statement to display the compression characteristic for the table, as shown in Example 4–46. Note that the SQL SHOW STORAGE MAP statement indicates whether compression is disabled for any table defined or imported with an explicit DISABLE clause.

Example 4–46 Using the SHOW Statement to Check Compression

```
SQL> SHOW STORAGE MAP EMPLOYEES_MAP
     EMPLOYEES_MAP
 For Table:            EMPLOYEES
 Placement Via Index:   EMPLOYEES_HASH
 Compression is:       DISABLED
 Store clause:         STORE USING (EMPLOYEE_ID)
             IN EMPIDS_LOW WITH LIMIT OF ('00200')
             IN EMPIDS_MID WITH LIMIT OF ('00400')
             OTHERWISE IN EMPIDS_OVER
SQL>
SQL> SHOW STORAGE MAP SALARY_HISTORY_MAP
     SALARY_HISTORY_MAP
 For Table:            SALARY_HISTORY
 Compression is:       DISABLED
 Store clause:         STORE IN SALARY_HISTORY
SQL>
SQL> SHOW STORAGE MAP JOB_HISTORY_MAP
     JOB_HISTORY
 For Table:            JOB_HISTORY
 Placement Via Index:   JOB_HISTORY_HASH
 Compression is:       ENABLED
 Store clause:         STORE USING (EMPLOYEE_ID)
             IN EMPIDS_LOW WITH LIMIT OF ('00200')
             IN EMPIDS_MID WITH LIMIT OF ('00400')
             OTHERWISE IN EMPIDS_OVER
```

4.3.3.2 Summary of Data Compression Options

The data compression options previously discussed are summarized as follows:

- ENABLE COMPRESSION (the default)

  Leave data compression enabled for the tables used most frequently by retrieval applications.

- DISABLE COMPRESSION

  Disable data compression for the tables that are updated frequently, and especially for the rows that are modified and retrieved often.

  This option must be explicitly stated in the SQL CREATE or ALTER STORAGE MAP statement or in the SQL IMPORT statement where a new storage map is defined for the first time with the SQL CREATE STORAGE MAP statement.

4.4 Adjusting OpenVMS Parameters for Oracle Rdb Applications

OpenVMS VAX OpenVMS Alpha

When you change operating system parameters, note the effect of one change before you make another. Try out operating system parameter changes when the system is not very busy or when these changes would cause only a minor disruption of business.

Schedule adequate down time for your database to make a change and verify if it helped, and to see if the change adversely affected something else. You must decide whether to tune your operating system and process quotas for database use, or whether you must balance this need against the needs of other users of computer resources.

In changing system and process parameters, you might find your time can be better spent investigating the details of your application design, once you have provided adequate memory, file limits, enqueue limit, byte limit, and so on. See the *Oracle Rdb7 Installation and Configuration Guide* for further information on how to set system and process parameters. Once you make sure operating system and user-process parameters are large enough, you may not gain much by continuing to tune the operating system. You may be more likely to achieve significant results elsewhere. Refer to Section 7.6.1 for additional information on tuning the system.

The OpenVMS Monitor utility is useful for monitoring system performance. This is a software sampling performance monitor used to obtain information on the system as it is running and to obtain summary reports of previous system activity. See the OpenVMS documentation set for more information on using the OpenVMS Monitor utility. You can also use the Performance Monitor VM Usage Statistics screen described in Section 4.4.1. ♦

4.4.1 Performance Monitor VM Usage Statistics Screen

The VM Usage Statistics screen shows a summary of the dynamic virtual memory usage for all database users on the VMScluster node. (Virtual memory is unique to each database user.) The following is an example of a VM Usage Statistics screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1996 13:38:09
Rate: 3.00 Seconds           VM Usage Statistics          Elapsed: 03:17:15.56
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------


statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

GET_VM calls                  1        0       1.0        12914        1844.8
FREE_VM calls                 0        0       0.3         3560         508.5

GET_VM kilobytes              0        0       0.4         4462         637.4
FREE_VM kilobytes             4        0       3.5        41393        5913.2

$EXPREG calls                 0        0       0.0            0           0.0



--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For information about each of the fields shown in this display, see the Performance Monitor help.

4.4.2 Checking and Setting OpenVMS System Parameters

OpenVMS VAX OpenVMS Alpha

You can use AUTOGEN to reset parameter values and system file sizes to meet the operational and performance characteristics your applications require. The new values and file sizes take effect the next time the system is booted. For more information on AUTOGEN, see the OpenVMS documentation covering system management tasks and the *Oracle Rdb7 Installation and Configuration Guide.*

System parameters you should check include the following:

- CHANNELCNT

  Set high: CHANNELCNT = (application dependent).

  CHANNELCNT specifies the number of permanent I/O channels available to the system. You should set this value to a number larger than the largest FILLM value in the database environment. One channel count is used every time a file is opened in your database application. For example, in a single-file database with one user, this value will be over 4: one for the .rdb file, one for the .snp file, one for the user's .ruj file, and one for the .aij file (when enabled). Temporary files such as sort files and files assigned to SYS$OUTPUT and SYS$INPUT also use channel count.

  Section 4.4.4 describes the FILLM quota.

- IRPCOUNT, IRPCOUNTV

Run AUTOGEN after your application is running and tune according to the feedback you receive.

Note

With OpenVMS VAX V6.0 and OpenVMS Alpha V1.5 and higher, the operating system sets these parameters automatically. The system manager does not need to set or modify these parameters.

IRPCOUNT sets the number of preallocated intermediate request packets (IRPs). Each IRP requires 160 bytes of permanent resident memory. If the IRPCOUNT value is too large, physical memory is wasted. If the IRPCOUNT value is too small, the system increases its value automatically, as needed, to permit proper performance. However, the system cannot increase IRPCOUNT beyond the value of IRPCOUNTV. If you check your system using the DCL command SHOW MEMORY/POOL/FULL and see that CURRENT IRPs in use is a value larger than the value for INITIAL IRPs, you should increase the SYSGEN parameter to at least the CURRENT IRP value.

IRPCOUNTV establishes the upper limit to which the IRPCOUNT value can be automatically increased by the system. If this parameter value is set too low, system performance can be adversely affected because IRPCOUNTV cannot be used for nonpaged pool requests.

- LRPCOUNT, LRPCOUNTV

  Run AUTOGEN after your application is running and tune according to the feedback you receive.

Note

With OpenVMS VAX V6.0 and OpenVMS Alpha V1.5 and higher, the operating system sets these parameters automatically. The system manager does not need to set or modify these parameters.

LRPCOUNT sets the number of preallocated large request packets (LRPs). Each LRP uses 576 bytes of permanent resident memory that is equal to the number of bytes specified by the LRPSIZE parameter. (Normally, LRPSIZE is 576 bytes.) If the LRPCOUNT value is too large, physical memory is wasted. If the LRPCOUNT value is too small, the system increases its value automatically, as needed, to permit the system to perform properly. However, the system cannot increase LRPCOUNT beyond the value of LRPCOUNTV. If you check your system using the DCL command SHOW MEMORY/POOL/FULL and see that CURRENT LRPs in use is a value larger than the value for INITIAL LRPs, you should increase the SYSGEN parameter to at least the CURRENT LRP value.

LRPCOUNTV establishes the upper limit to which the LRPCOUNT value can be automatically increased by the system. If this parameter value is set too low, system performance can be adversely affected by preventing the system from using this memory allocation mechanism for nonpaged pool requests.

- SRPCOUNT

  Set high: SRPCOUNT = 8192 (for small systems); = 15000 (for large systems).

  Note

  With OpenVMS VAX V6.0 and OpenVMS Alpha V1.5 and higher, the operating system sets this parameter and SRPCOUNTV automatically. The system manager does not need to set or modify these parameters.

  SRPCOUNT sets the number of preallocated small request packets (SRPs). Each SRP uses 96 bytes of permanent memory. Oracle Rdb uses one SRP (or 96 bytes) for each lock. Never set the SRPCOUNT value lower than the value for LOCKIDTBL because each lock uses one SRPCOUNT. If you check your system using the DCL command SHOW MEMORY/POOL /FULL and see CURRENT SRPs in use is a value larger than the value for INITIAL SRPs, you should increase the SYSGEN parameter to at least the CURRENT SRP value.

  When global buffers are enabled, the number of locks used increases, which may require that you increase the SRPCOUNT value.

- SRPCOUNTV

  Set high: SRPCOUNTV = 32768 (for small systems); = 60000 (for large systems).

  SRPCOUNTV establishes the upper limit to which the SRPCOUNT value can be increased. If you set the SRPCOUNT value manually using SYSGEN, set SRPCOUNTV to the preceding values (four times the SRPCOUNT value). If AUTOGEN was used to set the SRPCOUNT value for your system, do not explicitly set a value for SRPCOUNTV, because OpenVMS always sets this properly (usually four times the SRPCOUNT value).

When global buffers are enabled, the number of locks used increases, which may require that you increase the SRPCOUNTV value.

- LOCKIDTBL

  Set high: LOCKIDTBL = 8192.

  LOCKIDTBL sets the initial number of entries in the system Lock ID table. It also determines the amount by which the Lock ID table will extend if the system runs out of locks. The system will automatically extend the Lock ID in increments of the SYSGEN parameter LOCKIDTBL when it runs out of space. There is one entry in the Lock ID table for each lock in the system, and each entry requires 4 bytes. Whenever you change the value of LOCKIDTBL, you should examine the value of REHASHTBL and change it, if necessary.

  You can monitor locks with the Performance Monitor or with the OpenVMS Monitor utility (MONITOR).

  LOCKIDTBL is allocated from the nonpaged pool. If you set this parameter too low, programs can receive the following error message:

  ```
  %SYSTEM-E-NOLOCKID, no lock id available
  ```

  When global buffers are enabled, the number of locks used increases, which may require that you increase the LOCKIDTBL value.

- LOCKIDTBL_MAX

  Set high: LOCKIDTBL_MAX = a value greater than LOCKIDTBL

  LOCKIDTBL_MAX specifies an upper limit on the size of the Lock ID table. This dynamic system parameter must be set permanently to a value equal to or greater than 2048. If you set LOCKIDTBL to 8192, then LOCKIDTBL_MAX should be set to a value greater than 8192. Do not lower this value after you install Oracle Rdb. If you set this parameter too low, programs can receive the following error message:

  ```
  %SYSTEM-E-NOLOCKID, no lock id available
  ```

  When global buffers are enabled, the number of locks used increases, which may require that you increase the LOCKIDTBL_MAX value, based on the total number of global buffers.

- NPAGEDYN

  Run AUTOGEN after your application is running and tune according to the feedback you receive.

NPAGEDYN defines the size of the nonpaged dynamic pool in bytes. This parameter NPAGEDYN establishes the initial setting of the nonpaged pool size, but the pool size can be increased dynamically. To set a value for this parameter, use the default value initially, and then monitor the amount of space actually used with the DCL command SHOW MEMORY/POOL /FULL.

- NPAGEVIR

  Run AUTOGEN after your application is running and tune according to the feedback you receive.

  NPAGEVIR defines the maximum size to which NPAGEDYN can be increased. If this value is too small, systems could hang. To set this parameter, use the default value initially, and then monitor the amount of space actually used with the DCL command SHOW MEMORY/POOL /FULL.

- GBLPAGES

  The required minimum value for GBLPAGES is *n* plus 1396 pages, where *n* refers to the system parameter value that is in use prior to installing Oracle Rdb.

  GBLPAGES sets the number of global page table entries allocated at startup time. Each global section requires one global page table entry. Every 128 entries add 4 bytes to permanently resident memory in the form of a system page table entry.

  When global buffers are enabled for a database, the size of the global section increases, which could mean that you need to increase the GBLPAGES value. See Section 4.1.2.12 for more information.

  Note

  As you increase the value of GBLPAGES beyond its default setting, you must adjust SYSMWCNT. For every 128 pages you add to GBLPAGES, increase SYSMWCNT by 1. SYSMWCNT sets the quota for the size of the system working set, the paged dynamic pool, OpenVMS RMS, and the resident portion of the system message file. While a high value for SYSMWCNT takes space away from the user working sets, a low value may seriously impair system performance. When you use AUTOGEN, these parameters are set automatically.

- GBLSECTIONS

The required minimum value for GBLSECTIONS is *n* plus 80 sections, where *n* refers to the system parameter value that is in use prior to installing Oracle Rdb.

GBLSECTIONS sets the number of global section descriptors allocated in the system header at startup time. Each section takes 32 bytes of permanently resident memory.

- MAXBUF

  The required minimum value for MAXBUF is 1200 bytes.

  MAXBUF sets the maximum size in bytes of a buffered I/O transfer. Space for buffered I/O transfers is allocated from the permanently resident nonpaged dynamic pool.

- PROCSECTCNT

  The required minimum value for PROCSECTCNT is 32 sections.

  PROCSECTCNT sets the number of section descriptors that a process can contain. Each section descriptor increases the fixed portion of the process header by 32 bytes. You should set this value greater than the maximum number of image sections in any section to be run, as indicated by the linkage memory allocation map for the image.

- VIRTUALPAGECNT

  Increase the current value for VIRTUALPAGECNT to 2000 additional pages for each active database.

  VIRTUALPAGECNT sets the maximum number of virtual pages that can be mapped for any one process. Every 128 virtual pages requires 4 bytes of permanently resident memory in the system page table.

  When global buffers are enabled for a database, the size of the global section increases. Because your process maps to the global section, as the global section grows, so must the number of virtual pages you are allowed to have. Therefore, an increase in the size of the global section could mean that you need to increase the VIRTUALPAGECNT value. See Section 4.1.2.12 for more information.

- WSMAX

  Set high: WSMAX = set to largest working set size needed on the system.

  WSMAX sets the maximum number of pages on a systemwide basis for any working set. Set the value for WSMAX to the size of the largest working set needed on your system. This is useful in a heterogenous cluster environment, where memory differs but a common UAF file is used. The default value is appropriate for normal time-sharing operations, while significantly larger values should be used only to reduce page faulting for programs with very large virtual address spaces.

- DEADLOCK_WAIT

  DEADLOCK_WAIT defines the number of seconds a lock request must wait before the system initiates a deadlock search.

  Set DEADLOCK_WAIT low for a multiuser application involving a high-contention, high-update work load in either a VMScluster or noncluster environment. One second per node in the VMScluster +1 is recommended. If 10 nodes are in the VMScluster, set DEADLOCK_WAIT to 11 seconds. When DEADLOCK_WAIT is set to 11 seconds, a lock request must wait 11 seconds before the system initiates a deadlock search.

  Work loads characterized by read-only tasks use fewer lock resources and are not likely to encounter deadlocks. Therefore, a low value for DEADLOCK_WAIT in a read-intensive environment is not as critical and might decrease overall performance by causing unnecessary checking for deadlocks. In this case, the default value of 10 is recommended.

  You may decide to set different values for this parameter and determine which value best meets the performance needs for your particular database environment. Monitor lock statistics on your system using MONITOR. This utility provides information on both deadlock search and deadlock find counts. Oracle Corporation recommends that you leave the default value of 10 seconds for DEADLOCK_WAIT unless MONITOR shows a high percentage of deadlocks. Set the value to less than 10 seconds to get deadlocks signaled more frequently. If, at 10 seconds, there are frequently no deadlocks found, set the DEADLOCK_WAIT value to 20 or 30 seconds. Be careful when you use the Performance Monitor to tune DEADLOCK_WAIT because this command shows you locking statistics for just the database in question, not the entire system.

  See Section 8.4 for additional information on the DEADLOCK_WAIT parameter. ♦

OpenVMS
Alpha ☰

- VCC_FLAGS, VCC_MAXSIZE

  OpenVMS Alpha allows you to use a virtual I/O cache on Alpha systems to reduce I/O bottlenecks and improve performance. The virtual I/O cache works only on single-node systems; clustered systems do not use a virtual I/O cache.

  After upgrading to OpenVMS Alpha Version 1.5, you may notice a significant number of swapped out processes when you use the SHOW SYSTEM command. You might also notice that resource-intensive database operations, such as creating new databases, return SYSTEM-W-POOLEXPF errors to the operator console, or fail altogether with

INSFMEM errors. These problems are not specific to Oracle Rdb; other resource-intensive applications may cause the same symptoms to occur.

The source of these problems may be that the OpenVMS Alpha Version 1.5 upgrade procedure has set the virtual I/O cache size to a very large value, which in turn requires a significant amount of physical memory to support.

You can determine whether or not this is the problem by using SYSGEN to check the SYSGEN parameters VCC_FLAGS and VCC_MAXSIZE. The VCC_FLAGS parameter is used to enable or disable virtual I/O caching. Use SYSGEN to see if the virtual I/O cache is enabled:

```
SYSGEN>  SHO VCC_FLAGS
Parameter Name            Current    Default     Min.     Max.     Unit  Dynamic
--------------            -------    -------    -------  -------   ----  -------
VCC_FLAGS                       1          1         0        -1 Bitmask
```

If VCC_FLAGS is set to 1, the cache is enabled (a value of 0 means the cache is disabled). If the cache is enabled, check the VCC_MAXSIZE parameter:

```
SYSGEN>  SHO VCC_MAXSIZE
Parameter Name            Current    Default     Min.     Max.     Unit  Dynamic
--------------            -------    -------    -------  -------   ----  -------
VCC_MAXSIZE            2000000000       6400           02000000000 Blocks
```

By default, memory is allocated for caching 6400 disk blocks. This requires 3.2 megabytes of memory. If you have a very large value set for the current VCC_MAXSIZE value, reduce it to 6400 blocks and reboot your system. This will correct the problem. ♦

4.4.3 Tuning Working Set Adjustment Parameters

OpenVMS VAX OpenVMS Alpha

Memory management involves tuning working set adjustment parameters. The goal of this task is to maintain an even balance between memory use and the number of processes that can run concurrently on the system that require physical space in physical memory. The physical memory space allotted to each process is known as the **working set**.

In general, the working set consists of all the valid pages in memory for any particular process. The working set usually represents a subset of the total number of pages in the process' page tables. Pages are kept in the working set or in the page cache, or on the disk for each process in image, section, paging, and swapping files. All processes have an initial working set limit of pages known as the working set default, WSDEFAULT. As the WSDEFAULT limit is used up, each process can expand to a quota of pages known as the working set quota, WSQUOTA, through the automatic working set adjustment (AWSA) feature. A process will increase from WSDEFAULT to WSQUOTA without checking to see if any memory is available. For processes that need even more pages, pages can be increased to a maximum working set limit known as the working set extent, WSEXTENT. The maximum working set limit that can be assigned to any program is always a value greater than WSEXTENT and is known as the system parameter working set maximum, WSMAX.

The memory management strategy depends initially on the limits in effect for WSQUOTA and WSEXTENT for each process. For a process, these limits are derived from the user authorization file (UAF) as assigned by the system manager or determined from the DEFAULT record and assigned by the system. Care should be given to the limits you assign to WSDEFAULT, WSQUOTA, and WSEXTENT for each user. Limits should be selected for WSQUOTA that are large enough to allow each user's process to perform reasonably well without requesting additional pages, yet remain small enough so that a single process is not guaranteed an inequitable share of memory when memory is limited. Generally, the most desirable working set limit lies just above the point where performance drops sharply. Therefore, set these initial limits for the working set for different types of processes based on a strategy that considers the desired automatic working set adjustment. By adopting this type of strategy, you will know when the parameters are out of adjustment and how to direct your tuning efforts. See the OpenVMS documentation set for detailed guidelines on setting initial working set values for tuning automatic working set adjustment parameters for two general strategies:

- Tuning for rapid response times whenever the workload demands greater working set sizes in an ever-changing, time-sharing environment
- Tuning for less dynamic response times that will stabilize and track moderate needs for working set growth in a production environment ♦

4.4.4 Checking and Setting User Account Parameters

OpenVMS VAX OpenVMS Alpha

The values suggested in this section are *minimum* settings; the settings required by users on your system might differ substantially. The suggested values are specific only to the use of Oracle Rdb. You should add the values required for other OpenVMS layered products to the value you choose to use for Oracle Rdb and modify the values for each user as needed.

See the OpenVMS documentation set for information on how to use the AUTHORIZE utility and UAF records. The UAF parameters pertinent to Oracle Rdb users are:

- ENQLM

  The lock queue limit, which is the maximum number of locks that can be queued at any one time.

  Set high: 2000.

You may need to set ENQLM higher for ad hoc use, if you are using the repository, or if you have enabled global buffers. When global buffers are enabled for a database, one lock is used for each page that a user has in his or her allocate set, and each of these locks is charged against ENQLM. ENQLM limits the number of locks a process can own.

- WSDEFAULT, WSQUOTA, WSEXTENT

  The default working set size, the working set quota, and the working set extent.

  WSDEFAULT sets the initial working set size limit for a user's process. WSQUOTA guarantees the user that the number of physical pages specified will be available. WSEXTENT sets the maximum number of pages on a systemwide basis for any working set. If the automatic working set adjustment (AWSA) feature is enabled, it can affect the user's working set size.

  Set high: WSDEFAULT 512; WSQUOTA 1024 (depending on available memory); WSEXTENT 2048 or higher.

  If the database will be used for many join operations, a bigger page size will probably be required to process the join requests more efficiently. If the database has enabled global buffers and has a large number of global buffers, you may need higher quotas for WSDEFAULT, WSQUOTA, and WSEXTENT. See the OpenVMS documentation set for a more detailed discussion on this topic and guidelines on setting initial working set values for tuning automatic working set adjustment parameters.

  If you perform any explicit sorting operations that include projection (SQL ORDER BY and DISTINCT) or an implicit sort operation as performed by the query optimizer, and the query takes more time than expected to complete and you notice lots of disk accesses, check your WSEXTENT and WSQUOTA parameter values to see if they are set properly for the operation. For example, if the WSEXTENT value is 42,000 pages and the WSQUOTA value is 20,000 pages, try setting the WSEXTENT parameter value closer to the value for WSQUOTA. That is, WSEXTENT=WSQUOTA+2000. The Sort Utility (SORT) (which is called many times during this query execution) uses the difference between these two parameters (in this case 2000 pages) to allocate VM for scratch space. This results in excessive paging when a disk-based temporary file would be more efficient.

  Sometimes less memory is desirable for these kinds of operations. If the system is heavily used, the OpenVMS operating system may be unable to allocate all the pages in the working set extent to your process. To avoid excessive paging in a heavily used system, make the working set extent lower. See DCL help for more information.

- FILLM

  The open file limit, which specifies the maximum number of files that a user's process can have open at a time.

  Set high: 100 for OpenVMS V5.4; 100 for OpenVMS V5.5; 300 for OpenVMS V6.0

  Each database storage area, snapshot file, .ruj file, .aij file, root file, or log file can be opened by the user. Other application-specific files (such as images and data files) or access to the database through the repository add to the total number of open files. Each open file is charged against the user's file limit.

  For a multifile database, set this value to the sum of the .rdb, .rda, .snp, .ruj, .aij, and the temporary work files for the largest application running on your system. Use the .ruj file value for the number of users (actually the number of attaches to the database) for the application that creates the largest number of .ruj files. Use the RMU Dump Users command to determine this value.

  If the FILLM quota is exceeded, the current operation aborts and an exceeded quota message is returned to the user. Increasing the FILLM quota requires a corresponding increase to the SYSGEN parameter CHANNELCNT if the new FILLM quota is higher than the current value of CHANNELCNT.

- BYTLM

  The buffered I/O byte limit, which is the maximum number of bytes of nonpaged system dynamic memory that can be specified at one time by a user's job for transfer to outstanding buffered I/O operations.

  Set high: 20480 or more.

- ASTLM

  The asynchronous trap limit, which is a limit on the number of outstanding ASTs for a process.

  Use the following formula to determine a value for ASTLM:

  ```
  ASTLM > DIOLM + 6 (or equals 24 if DIOLM is set to 18)
  ```

The AST queue limit is the maximum number of AST operations and scheduled wake-up requests that can be outstanding at any one time. The ASTLM value should be greater than the value for DIOLM, plus 6. The DIOLM value should be equal to or greater than the value for the defined number of database buffers. The database buffers are written back to the database in parallel. Therefore, there might be an outstanding AST for each buffer.

See the following description of DIOLM for information about displaying the value of the database's number of buffers parameter.

- DIOLM

  Direct I/O count limit is the count limit or maximum number of direct I/O operations (usually disk) that can be outstanding at one time.

  Set this value initially to 18.

  For improved performance, the DIOLM value should be equal to or greater than the value for the defined number of database buffers when local buffers are enabled for a database. Use the Performance Monitor Buffer Information screen to get the value of the database's number of buffers parameter.

  For a process that will access the mf_personnel database, a DIOLM value of 20 and an ASTLM value of 26 should be sufficient.

  When a database has global buffers enabled, the DIOLM value should be equal to or greater than the value for the USER LIMIT parameter (the largest allocate set allowed for any user in the database). The RMU Show Users command displays the active value of the USER LIMIT parameter for a database on the current node, as shown in the following example:

  ```
  $ RMU/SHOW USERS mf_personnel.rdb
      .
      .
  - maximum global buffer count per user is 25
  ```

  For a process that will access the mf_personnel database and use global buffers, a DIOLM value of 25 and an ASTLM value of 31 should be sufficient.

- BIOLM

  Buffered I/O limit count, which is the maximum number of buffered I/O operations that can be outstanding at one time. Set this value initially to 18.

  Set BIOLM to the same value as DIOLM. Adjust BIOLM whenever you modify DIOLM.

- PRCLM

  The Subprocess creation limit, which is the maximum number of subprocesses that can exist at one time for the user’s process.

  This value can be set to 1.

- PGFLQUOTA

  Paging file limit, which is the maximum number of pages that the user’s process can use in the system paging file.

  This value should be set to a minimum value of 20,000. ♦

Volume II

5

The Query Optimizer

The relational database model represents the user's view of data stored in a database. Consequently, determining the most efficient way to retrieve that data can be a very complex task. Oracle Rdb contains a **query optimizer** (frequently referred to as the optimizer) that automatically analyzes every query to determine the most efficient method of data access.

5.1 Optimizer Responsibilities

Database performance is directly affected by the ability of a database to access a row or rows stored on disk through I/O operations. The greater the number of I/O operations, the longer it takes to find and retrieve rows that satisfy a query. To keep I/O operations to a minimum, Oracle Rdb uses the query optimizer. The optimizer's responsibility is to determine the most I/O efficient method of retrieving user-requested data, and to establish access paths to that data. Although the optimizer may consider different retrieval possibilities, all retrieval strategies yield the same requested data.

A significant portion of optimization is performed during query compilation. This **static** optimization is done only once before any query execution takes place. The primary goal of the static optimizer is to deliver the query execution strategy that requires the fewest number of I/Os compared to other possible strategies. A selection criterion for the best strategy is a pure *mathematical* minimum execution cost, which is calculated using rough estimates of factors such as table and index cardinality, workload and storage statistics, and assumptions of how any relational operators typically change the incoming data quantity and distribution. These basic estimates and assumptions often prove effective, but in some cases they can be inaccurate and lead the query optimizer to select a strategy that is less than optimal.

To compensate for potential static optimizer estimation errors, the Oracle Rdb query executor uses a **dynamic** optimizer that controls the query retrieval process and can dynamically switch to a better strategy. A between-strategy switch occurs when the cost estimate of a strategy, originally rejected by the optimizer in favor of the current strategy, improves during query execution to the point that it provides a more economical solution. Dynamic optimization imitates the human way of solving a retrieval problem by recognizing that estimates are not precise, and by using a powerful set of tools that try to prevent mistakes but resolve any mistakes that do occur.

For most straightforward and simple queries, the static optimizer selects the most efficient execution plan. If the selected strategy does prove less than optimal at the single table retrieval level, the dynamic optimizer can revise and correct that strategy during execution. However, there still may be occasions when you feel the selected solution is not the best. This chapter provides an overview of the optimizer, explains the different access methods used to optimize a query, and describes what you can do to influence the optimizer. Appendix C provides examples of the access methods, and describes how to use the logical name RDMS$DEBUG_FLAGS and the configuration parameter RDB_DEBUG_FLAGS to examine query costs, optimizer strategy, and optimizer execution.

5.2 Optimizer Terminology

There are several terms, specific to the Oracle Rdb query optimizer, that need to be defined immediately. Sections 5.2.1 through 5.2.3 define these terms. Other query optimizer terms are defined as they are introduced.

5.2.1 Predicate Selectivity

Predicate selectivity is the estimated fraction of total rows in a table or in an intermediate result table for which the predicate is true. Predicate selectivity depends *only* on the operator, and not on the operands (that is, the data or values in the query). For example, the optimizer estimates the selectivity of X > 10, X > 125, or Y > 70 to be 0.35 in all three cases because all three cases use the operator ">." Predicate selectivity is applied at the earliest possible time in query optimization to reduce the total number of rows as soon as possible.

5.2.2 Strategy

Strategy refers to a sequence of operations performed on data to solve a query and produce a result. A strategy is produced by the optimizer and represents an optimal solution for a query. An optimal solution is the one that the optimizer estimates will require the minimum number of disk I/Os.

5.2.3 Cost

Cost is the estimated number of disk I/O operations that a query solution will require. This is the metric used by the optimizer to identify an optimal solution—one with minimum cost. The cost of a solution is based on the estimated cardinality and the type of access method the optimizer is investigating. Factors affecting the cost include the following:

- Query structure
- Query predicates
- Result ordering or grouping
- Type of retrieval used
- Type of join used (if multiple table access)
- Type of storage area (uniform or mixed)
- The page size for the storage area (see Section 5.2.3.1)
- Cardinality statistics
- Workload statistics (if collected)
- Storage statistics (if collected)

Examine the cost of a query by using the logical name RDMS$DEBUG_FLAGS or the configuration parameter RDB_DEBUG_FLAGS. Refer to Appendix C for information.

5.2.3.1 How the Optimizer Estimates Page Size for Tables That Store Data in Multiple Storage Areas

Oracle Rdb allows you to define a storage map that specifies the storage area in which table rows will be stored, based on the value of one or more columns in the table. For example, the following storage map definition specifies that all rows in the table TABLE1 with a value not greater than 200 for the column COLUMN1 will be stored in storage area AREA1 and all other rows from TABLE1 will be stored in storage area AREA2:

```
SQL> CREATE STORAGE MAP MAP1 FOR TABLE1
cont> STORE USING (COLUMN1)
cont>    IN AREA1 WITH LIMIT OF ('200')
cont>    OTHERWISE IN AREA2;
```

It is possible for the two storage areas (AREA1 and AREA2) to have different page sizes. Whenever a storage map for a table specifies that rows for the table will be stored in multiple storage areas, the optimizer uses the page size of the first area (as it appears in the storage map definition) for all estimations on the table, regardless of which area is used for retrieval.

5.3 Optimizer Statistics

The optimizer uses statistics to determine the cost of a solution. Three types of statistics available to the optimizer to select a strategy include:

- Cardinality
- Workload
- Storage

Oracle Rdb automatically maintains cardinality statistics. Workload and storage statistics have to be collected using the RMU Collect Optimizer_Statistics command.

See the *Oracle RMU Reference Manual* for more information on collecting and maintaining optimizer statistics.

5.3.1 Cardinality Statistics

The **cardinality statistics** capture data volume information. Cardinality refers to quantity. The cardinality statistics include table cardinality, index cardinality, and index prefix cardinality.

5.3.1.1 Table Cardinality

Table cardinality specifies the number of rows in a table. Each retrieval solution (both intermediate and final) also has a cardinality. The cardinality of a complete solution is the estimated number of rows that a query will return.

The approximate cardinality of a table is stored in the field RDB$CARDINALITY in the system table RDB$RELATIONS. This provides all database users with a single location from which to retrieve a table's approximate cardinality.

When a table is first referenced by a process, the value in RDB$CARDINALITY is loaded in the symbol table associated with that process. This can cause problems if the cardinality of the table is extremely volatile or if the RDB$SYSTEM storage area is set to read-only. Oracle Rdb has no way to update each process' copy of RDB$CARDINALITY if there is a significant change. RDB$CARDINALITY is not updated every time a user adds or deletes a row from a table. Instead, statistics are kept for each user on how many rows were added or deleted. Then, at commit time, if the total number of changes exceeds $\log_2$ of the current cardinality, RDB$CARDINALITY is updated. Otherwise, RDB$CARDINALITY is updated when a process detaches. The more rows there are in a table, the less chance there is that RDB$CARDINALITY will be sufficiently different to affect the optimizer strategy.

5.3.1.2 Index Cardinality

In addition to table cardinality, Oracle Rdb keeps track of the cardinality of each sorted index with duplicates allowed in the database. The cardinality of such an index is defined as the number of distinct values in that index. Only the full index cardinality is stored, not subsets as in a multisegmented index.

Oracle Rdb maintains cardinality for each index that allows duplicates so that the optimizer can reference a count of how many unique values occur in the index. Consequently, the optimizer can prioritize the suitability of two or more indexes based on which has the highest number of unique values.

For example, an index on SEX would have a cardinality of 2, while an index on LAST_NAME in a table of 1000 rows might have a cardinality of 700. The optimizer can distinguish between these two indexes and will use the one with the higher cardinality value if there is an equality restriction on the index column. Index cardinality maintenance adds a small amount of overhead to update transactions that modify indexed columns, and to those transactions that store or delete rows. Unlike non-unique indexes, the overhead is very low on unique indexes because Oracle Rdb does not maintain the unique index cardinality; instead, the optimizer uses table cardinality as the cardinality of a unique index to estimate index cost.

Index cardinality is stored in the RDB$CARDINALITY field in the RDB$INDICES system table.

5.3.1.3 Index Prefix Cardinality

Index prefix cardinality is the number of distinct key values in leading parts of a multisegmented sorted index. That is, the number of distinct values in the first segment alone, the number of distinct values in the first and second segments combined, the number of distinct values in the first, second, and third segments combined, and so on. This statistic is meaningful only for sorted indexes; therefore, it is not maintained for hashed indexes.

The optimizer uses index prefix cardinality to determine the portion of the B-tree index that needs to be scanned. This allows the optimizer to estimate the cost of index retrieval. Index prefix cardinality is also used to estimate the number of rows fetched from a table.

Index prefix cardinality is stored in the RDB$CARDINALITY field in the RDB$INDEX_SEGMENTS system table.

5.3.1.4 Correcting Table and Index Cardinality

Because it is possible for the values stored in the system tables to differ from the actual number of table rows or index keys, the optimizer can create retrieval strategies based on obsolete cardinality values.

You can correct cardinality with the RMU Collect Optimizer_Statistics command. For example, to correct table and index cardinality for the EMPLOYEES table, enter the following command:

```
$ RMU/COLLECT OPTIMIZER_STATISTICS mf_personnel/STATS=(CARDINALITY) -
_$ /TABLE=(EMPLOYEES)/LOG
```

```
$ rmu -collect optimizer_statistics mf_personnel \
> -statistics=\(cardinality\) -tables=\(EMPLOYEES\) -log
```

Note that this command also updates index prefix cardinality if multisegmented sorted indexes are defined on the EMPLOYEES table.

The cardinality of a unique index may be altered using the RMU Collect Optimizer_Statistics command. However, Oracle Rdb does not make use of the stored value, nor will it attempt to update the value as rows are stored or deleted.

5.3.2 Workload Statistics

The **workload statistics** capture data distribution information in the form of duplicity and null factors. The duplicity and null factors are collected for each workload column group. A workload column group contains one or more columns from a table. The workload column groups are identified by the optimizer based on a query workload. The workload column groups are identified from equiselections, equijoins, GROUP BY clauses, and DISTINCT clauses specified in each query of a workload. Each workload column group is stored in the RDB$WORKLOAD system table. The collection of workload column groups in RDB$WORKLOAD is referred to as a **workload profile**.

You must generate a query workload profile before you can collect workload statistics. Use the WORKLOAD COLLECTION clause of the SQL ALTER DATABASE or CREATE DATABASE statement to generate a query workload profile. In turn, Oracle Rdb creates an RDB$WORKLOAD system table.

Consider disabling workload collection after all or most of the query workload has been profiled. This will eliminate accidental creation of new workload column groups based on ad hoc queries that are not part of the regular workload.

After the workload profile has been collected, use the RMU Collect Optimizer_ Statistics command to collect duplicity and null factors for each workload column group. The duplicity and null factors are stored in the corresponding workload column group entry in the RDB$WORKLOAD system table.

The workload statistics help the optimizer to estimate solution costs and cardinalities with higher accuracy. This normally results in optimal processing of various queries in a workload.

When the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as O, the following line indicates the use of workload statistics in optimizing the query:

```
Workload and Storage statistics used
```

5.3.2.1 Column Duplicity Factor

The column duplicity factor is a ratio of the total number of rows to the number of distinct values in a workload column group. It is the arithmetic mean of the duplicates per distinct value of a workload column group. It is a single-point statistic about the data distribution in a workload column group.

This statistic is used in determining the selectivity factor of equiselections as well as cardinality of the result from equijoin or grouped aggregation or projection operations. When you group, you want to know how many groups are formed; when you project (duplicate elimination), you want to determine the number of distinct result rows; when you apply equiselection on a column or group of columns, you need to know on average how many rows will be selected; and when you perform equijoins on a column or group of columns, you need to know the fanout factor of the join attribute to determine cardinality of the joined result.

The column duplicity factor is stored in the RDB$DUPLICITY_FACTOR field in the RDB$WORKLOAD system table.

This statistic is not automatically maintained by Oracle Rdb. Use the RMU Collect Optimizer_Statistics command to collect it.

5.3.2.2 Column Null Factor

The column null factor is a ratio of the number of rows with NULL value in at least one column of a workload column group to the total number of rows in a table. Since equijoins and equiselections imply removal of rows with NULL values, the column null factor is used to determine the number of rows that do not participate in an equiselection or equijoin operation. The column null factor is also used to estimate the cardinality of an outer join result.

The column null factor is stored in the RDB$NULL_FACTOR field in the RDB$WORKLOAD system table.

This statistic is not automatically maintained by Oracle Rdb. Use the RMU Collect Optimizer_Statistics command to collect it.

5.3.3 Storage Statistics

The **storage statistics** capture data clustering information such as the index key clustering factor, index data clustering factor, and table row clustering factor.

When the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as O, the following line indicates the use of storage statistics in optimizing the query:

```
Workload and Storage statistics used
```

5.3.3.1 Index Key Clustering Factor

The index key clustering factor is the average number of I/O operations needed to fetch a single index key and all dbkeys associated with it. It is a measure of how well an index key and its associated dbkeys are co-located. For a sorted index, it signifies the number of I/O operations needed to perform a full or range scan without row fetches. For a hashed index, it signifies the number of I/O operations needed to perform a hashed key lookup without row fetches. This statistic helps to improve the cost estimate of doing index-only retrieval using a sorted or hashed index.

The index key clustering factor is stored in the RDB$KEY_CLUSTER_FACTOR field in the RDB$INDICES system table.

This statistic is not automatically maintained by Oracle Rdb. Use the RMU Collect Optimizer_Statistics command to collect it.

5.3.3.2 Index Data Clustering Factor

The index data clustering factor is the average number of I/O operations needed to fetch all rows associated with a single index key. It is a measure of how well the index key order in a sorted index matches the physical placement order of data rows in storage areas. For a hashed index, it is a measure of how well all rows with the same key value are located in the same clump so that one I/O operation is sufficient to access the data.

The index data clustering factor reveals hidden data clustering possessed by an index. For example, a table may have been loaded with data that was already sorted by a key (a column or group of columns). If a sorted index is created using that key, the index will have full or close to full data clustering. If a table is loaded by placing rows via a hashed index, full or close to full data clustering will occur.

The index data clustering factor is used to estimate the cost of fetching data from a sorted index scan or from a hashed index lookup. Full data clustering results in a data fetch cost equal to the cost of sequentially fetching data rows for a sorted index scan, and equals 1 for hashed index lookup.

The index data clustering factor is stored in the RDB$DATA_CLUSTER_FACTOR field in the RDB$INDICES system table.

This statistic is not automatically maintained by Oracle Rdb. Use the RMU Collect Optimizer_Statistics command to collect it.

5.3.3.3 Table Row Clustering Factor

The table row clustering factor is the average number of I/O operations needed to sequentially fetch one row from a table. It is a measure of how well the data rows of a table are stored in different clumps of a storage area. This statistic also captures the effect of data row fragmentation, which may be due to row updates or due to rows that cannot fit into a single page and have to be split into several fragments. This statistic is used to estimate the cost of doing a sequential scan of a table.

The table row clustering factor is stored in the RDB$ROW_CLUSTER_FACTOR field in the RDB$RELATIONS system table.

This statistic is not automatically maintained by Oracle Rdb. Use the RMU Collect Optimizer_Statistics command to collect it.

5.3.4 Controlling the Collection of Workload and Storage Statistics

Using workload or storage statistics carries with it some risk. You can greatly reduce the potential for negative side effects by:

- Using query outlines for individual queries
- Deleting workload and storage statistics from the database
- Defining the RDMS$USE_OLD_COST_MODEL logical name or the RDB_USE_OLD_COST_MODEL configuration parameter

When you define RDMS$USE_OLD_COST_MODEL or RDB_USE_OLD_COST_MODEL to be any value, the optimizer does not use workload or storage statistics. Using RDMS$USE_OLD_COST_MODEL or RDB_USE_OLD_COST_MODEL allows you to:

- Test the optimization of a query workload with and without the use of storage and workload statistics
- Selectively disable the use of workload and storage statistics for particular users, processes, and batch jobs

When the RDMS$USE_OLD_COST_MODEL logical name or the RDB_USE_OLD_COST_MODEL configuration parameter is defined, the optimizer uses cost and cardinality functions that were in use prior to V7.0 and ignores any workload and storage statistics that have been collected.

Deassign the logical name or configuration parameter to enable the optimizer to start using workload and storage statistics again in cost and cardinality estimates.

5.4 Query Optimizer Overview

Depending on the complexity of the query, the optimizer uses a single table access method, or a combination of single table access methods and join methods, to arrive at a minimal cost solution for the query. To find a minimal cost solution, the optimizer creates solutions using different join orders and picks the one with the least cost. The number of join orders that the optimizer tries is directly related to the number of tables joined. If *n* tables are joined, then there are *n!* (factorial *n*) possible join orders. For example, for a 3-way join of tables A, B, and C, the optimizer tries 3! (which is equal to 6) join orders: ABC, ACB, BAC, BCA, CAB, CBA.

To solve a query, the query optimizer executes the following steps:

1. Determines a possible retrieval method for a table. This is based on the table columns specified in the query and the indexes defined on the table.
2. Estimates table cardinality; that is, how many rows would have to be accessed from the table. This estimate is based on the query predicates specified on the table.
3. Calculates the cost of the retrieval method. Cost is based on the estimated cardinality and the type of retrieval method the optimizer is investigating.
4. Compares the current retrieval method with other potential solutions already generated. Creates a new retrieval solution if any of the following conditions are true:
   - The cost of the current retrieval solution is less than previous retrieval solutions.
   - The current retrieval solution produces an interesting order, which subsequently may be useful to solve the entire query.
5. Determines a join method to join the current table to an existing intermediate solution that involves other tables if the query specifies two or more tables.

6. Creates a new join solution if any of the following conditions are true:
   - The cost of the current join solution is less than previous join solutions.
   - The current join produces an interesting order, which may be useful later on to solve the entire query.
7. If step 6 creates a new join solution, then repeat steps 1 through 6 to join each of the remaining tables and produce a complete solution for the query.
8. Repeats steps 1 through 7 for each of the join orders. Finally, picks a complete solution by selecting the solution with the minimum cost.

During step 4 and step 6, a previous solution is pruned if it is more costly than the current one and if it does not produce any interesting order. The optimizer will not create a complete solution using a join order if its partial solution becomes costlier than an existing complete solution. This early pruning of partial solutions significantly reduces the amount of time spent optimizing a query.

You do not have to be overly concerned about how to construct your queries; if your database changes, the query optimizer automatically devises a new access strategy. The order in which you specify joins, and the order of the clauses in a select expression do not, in most cases, influence the order the optimizer uses to satisfy a query. Refer to Section 5.8 for information on how to work with the optimizer.

5.5 Single Table Retrieval Methods

This section describes the strategies the optimizer can use to access a single table. In addition to accessing a single table, these strategies can be used as elements in a join to access rows in multiple tables.

You can use the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter to display the retrieval strategy for a query. Appendix C describes how to use the logical name and configuration parameter, and provides selected, annotated examples of access strategies.

The query optimizer can use the following methods for retrieving data from a single table:

- Sequential retrieval

  Accesses the database pages for a table's logical area sequentially, and reads all the rows in the table regardless of the selection expression used in the query. When RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as S, the following line indicates sequential retrieval:

```
Get   Retrieval sequentially of relation RELATION_NAME
```

- Dbkey retrieval

  Accesses table data directly through the dbkey (logical address) row pointer. Because this access path does not touch any index nodes, only the row itself is locked. When RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as S, the following line indicates dbkey retrieval:

  ```
  Get   Retrieval by DBK of relation RELATION_NAME
  ```

- Index retrieval

  Accesses a specific index structure (sorted or hashed) and retrieves the index keys, which include the dbkeys, of the rows. The optimizer then uses the dbkeys to fetch data rows. The data is delivered in index order if a sorted index is used. When RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as S, the following lines indicates index retrieval:

  ```
  Get   Retrieval by index of relation RELATION_NAME
  Index name INDEX_NAME
  ```

- Index-only retrieval

  Accesses only the index data. The selected index contains all the table columns specified in the query. Thus, no further row fetches are necessary. The optimizer delivers the data in index order if a sorted index is used. When RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as S, the following lines indicate index-only retrieval:

  ```
  Index only retrieval of relation RELATION_NAME
  Index name INDEX_NAME
  ```

  Index-only retrieval is possible with either sorted or hashed indexes.

- OR index retrieval

  Uses two or more sorted indexes, or a hashed index, defined on a single table when the predicates on these indexes are combined with the logical OR in the query. When RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as S, the following lines indicate OR index retrieval:

  ```
         OR index retrieval
           Get  Retrieval by index of relation RELATION_NAME
             Index name INDEX_NAME_1  . . .
           Conjunct  Get  Retrieval by index of relation RELATION_NAME
             Index name INDEX_NAME_2  . . .
  ```

The two indexes, INDEX_NAME_1 and INDEX_NAME_2, are processed separately as normal index retrievals and the rows fetched from these retrievals are concatenated. The optimizer discards any duplicate occurrences of the same row that may have been fetched from different OR legs. The data is not delivered in any particular order.

- Dynamic OR index retrieval

  Accesses a specific sorted index for a query that contains two or more predicates combined with the logical OR. The data is delivered in the index order. When RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as S, the following lines indicate dynamic OR index retrieval:

  ```
  Get  Retrieval by index of relation RELATION_NAME
  Index name  INDEX_NAME [1:1...]2
  ```

  See Section 5.7.1 for information on dynamic OR optimization.

- Dynamic leaf retrieval

  Chooses between indexes dynamically (during execution). When RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as S, the following line indicates dynamic leaf retrieval:

  ```
  Leaf#01  . . .
  ```

  See Section 5.7.2 for information on dynamic leaf optimization.

For single tables, the optimizer finds the cheapest retrieval solution by evaluating the cost for each possible retrieval strategy. The estimated retrieval cost for a solution includes the cost of scanning one or more useful indexes and the cost of fetching data records. The cheapest access path that produces rows in each interesting order and the cheapest access path that produces unordered rows are examined.

- If the query does not require a sort, the optimizer chooses the cheapest access path that produces unordered rows.
- If the query does require a sort, then the cost for producing that interesting order is compared with the cost for the cheapest unordered path added to the cost of sorting the rows into the proper order, and the cheapest path is selected.

The optimizer chooses index retrieval when a particular advantage is gained by using an index. If, however, the query does not specify a particular output order, the indexed columns are not used in the Boolean restriction specified for this table, and a given index cannot be used for retrieval without fetching the rows from a data area, there is no inherent reason to select an index. The only exception to this rule is when a query accesses rows in a mixed format storage area. The preferred strategy in this case might be to use an index, even if it requires retrieving all the rows. This is true because sequential retrieval would require reading the entire storage area (all rows in all tables in the area).

Key-Only Boolean Optimization

Key-only Boolean optimization is used in conjunction with index retrieval. Whenever possible, the optimizer uses this optimization to filter out as many index keys as possible before fetching rows. Thus the key-only Boolean optimization can save I/O operations by reducing the total number of row fetches.

To illustrate this optimization, assume a sorted index, EMP_FIRST_LAST, is defined using the FIRST_NAME and LAST_NAME columns of the EMPLOYEES table. The key-only Boolean optimization is used for the following query:

```
SQL> SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME
cont> FROM EMPLOYEES
cont> WHERE LAST_NAME STARTING WITH 'A'
cont> ORDER BY FIRST_NAME, LAST_NAME;

 EMPLOYEE_ID   FIRST_NAME   LAST_NAME
 00374         Leslie       Andriola
 00416         Louie        Ames
2 rows selected
```

If the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as S, the following strategy results from the previous query:

```
Conjunct        Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_FIRST_LAST [0:0] Bool
```

The optimizer uses the EMP_FIRST_LAST index because it provides the order specified in the query, and therefore there is no need to sort the result. The [0:0] notation indicates that all keys in the index are scanned, and the Bool notation means that a key-only Boolean is applied to each of these index keys. Here the optimizer uses the selection predicate, LAST_NAME STARTING WITH 'A', as a key-only Boolean. Note that the predicate LAST_NAME STARTING WITH 'A' cannot be used to scan only those keys whose value starts with 'A' because LAST_NAME is not a leading segment and there is no equality on the first segment.

The optimizer sequentially scans all keys in the EMP_FIRST_LAST index. The key-only Boolean is applied to each index key, and unwanted keys are filtered out. Then, the optimizer fetches only those rows that would satisfy the key-only Boolean. Without the key-only optimization, all rows from the EMPLOYEES table would have been fetched. This type of optimization generally saves a considerable number of I/O operations.

The optimizer also uses the key-only Boolean optimization for the following query:

```
SQL> SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME
cont> FROM EMPLOYEES
cont> WHERE FIRST_NAME LIKE '%G%' IGNORE CASE;
```

```
Leaf#01 FFirst EMPLOYEES Card=100
  BgrNdx1 EMP_FIRST_LAST [0:0] Bool Fan=9
```

```
 EMPLOYEE_ID   FIRST_NAME   LAST_NAME
 00175         George       Siciliano
 00319         Glenn        Silver
 00202         Margaret     Harrington
 00179         Meg          Dallas
 00267         Roger        Saninocencio
 00209         Roger        Smith
6 rows selected
```

For this query, the optimizer uses the predicate FIRST_NAME LIKE '%G%' IGNORE CASE as a key-only Boolean. The entire EMP_FIRST_LAST index is scanned, and the key-only Boolean is applied to each index key. The optimizer fetches only those rows whose keys satisfy the key-only Boolean. Out of 100 rows in the EMPLOYEES table, only 6 rows are retrieved.

Note that a LIKE predicate cannot be used to scan only those keys in the index that satisfy the predicate; instead, all keys in the index must be scanned. This is also true for CONTAINING predicates.

Min/Max Aggregate Optimization

The optimizer uses the min or max aggregate optimization when a query contains the MIN or MAX statistical functions. This optimization requires an appropriate sorted index, and the aggregate value must be located in a column that is part of the index. Other necessary conditions for this optimization include the following:

- The query should select a single MIN or MAX value.
- The column on which the value is based should be the first segment of the index. If the value is not based on the first segment, then all preceding segments should be based on equality selections. That is, the columns based on leading segments should equal some value or the columns should be null.
- The query selection should specify a single range for the index used.
- Selections should not be based on columns outside of the index.

- The sorted index to be used for min/max optimization should not have a VARCHAR or COLLATING SEQUENCE column as one of its segments.

The optimizer can use the min/max aggregate optimization with ascending, descending, partitioned, or multisegmented sorted indexes. When this optimization is selected, the optimizer performs a B-tree index descent to locate either the minimum or the maximum column value. This approach is much faster than either scanning a range of index keys or sequentially scanning all table rows to find the min or max column value.

The following sample queries use min/max optimization. The strategy output (generated by defining RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS as S) shows the notation indicating the min/max optimization. Refer to Appendix C for a complete description of the S flag notation.

```
SQL> SELECT MIN(EMPLOYEE_ID) FROM EMPLOYEES;

Aggregate       Index only retrieval of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [0:0]      Min key lookup

 00164
1 row selected
```

The notation “Min key lookup” indicates that the optimizer used the min/max optimization to find the minimum value for the EMPLOYEE_ID column.

```
SQL> SELECT MAX(EMPLOYEE_ID) FROM EMPLOYEES;

Aggregate       Index only retrieval of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [0:0]      Max key lookup

 00471
1 row selected
```

The notation “Max key lookup” indicates that the optimizer used the min/max optimization to find the maximum value for the EMPLOYEE_ID column.

```
SQL> SELECT MAX(EMPLOYEE_ID) FROM EMPLOYEES
cont>   WHERE EMPLOYEE_ID BETWEEN ’00200’ AND ’00300’;

Aggregate       Index only retrieval of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Max key lookup

 00287
1 row selected
```

In this example, the optimizer used the min/max optimization to find the maximum value for the EMPLOYEE_ID column and found this maximum column value within an index range (BETWEEN ’00200’ and ’00300’). The notation “[1:1]” indicates the existence of an index range.

The following example creates an index with two segments and then uses a query to select the maximum value from the column on which the second segment of the index is based:

```
SQL> CREATE INDEX SH_SALEND_SALARY
cont>   ON SALARY_HISTORY (SALARY_END, SALARY_AMOUNT);
SQL> --
SQL> -- Find the highest current salary.
SQL> SELECT MAX(SALARY_AMOUNT)
cont>   FROM SALARY_HISTORY
cont>   WHERE SALARY_END IS NULL;

Aggregate       Index only retrieval of relation SALARY_HISTORY
  Index name  SH_SALEND_SALARY [1:1]     Max key lookup

      93340.00
1 row selected
```

The keys in the index SH_SALEND_SALARY include two columns: SALARY_END and SALARY_AMOUNT. The optimizer uses the min/max optimization to find the maximum SALARY_AMOUNT value within the range of index keys that all have a null SALARY_END value.

5.6 Multiple Table Access Strategies

The degree of complexity in solving (optimizing) a query, and therefore the time required to produce a strategy, depends on the number of tables involved and the number of indexes on each table. To deal with this complexity, the optimizer joins one table at a time to an existing intermediate solution until a complete solution is created. The data source (that is, the record source stream or RSS) can be a table, a subquery solution, or a view.

There are three different methods for combining multiple data sources:

- Cross join
- Match join
- Merge

These strategies can be nested and, at the leaf level of nesting, each uses the single table access methods described in Section 5.5. The cross and match strategies join rows from two or more data sources, whereas the merge strategy (used for unions) concatenates rows from two or more data sources.

5.6.1 Cross Join

The cross join method, also known as the nested loop join, joins two or more data streams. Each data stream corresponds to an **Entry** in the cross join notation. A cross join method does not require that rows in any entry be in sorted order. When the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as S, the following lines within the query strategy indicate the cross join:

```
Cross block of 2 entries
   Cross block entry 1
         .
         .
         .
   Cross block entry 2
         .
         .
         .
```

The following steps are performed in a cross join:

1. A row is retrieved from the data stream, which can be a single table, in entry 1.
2. Any existing restriction predicate is applied to the row.
3. If the row satisfies the restriction predicate, all rows that match the join predicate are retrieved from the data stream in entry 2, one row at a time.
4. Steps 1 through 3 are repeated for each row retrieved from entry 1.

Generally, in a cross join with *n* entries, for each processed row from entry K, all rows matching a join predicate are retrieved from entry K+1. Then, for each processed row from entry K+1, all rows matching a join predicate are retrieved from entry K+2, and so on.

Entries following entry 1 often use index retrieval, because this is the most efficient way to find all matching rows based on a join predicate. Entry 1 can also use index retrieval if the optimizer finds an appropriate index. Because entry 2 is processed for each row in entry 1, the data stream with the least estimated cardinality is usually placed in entry 1, so that the number of iterations for processing subsequent entries is reduced.

In addition to the inner join, the cross join algorithm can also perform a left outer join. The right outer join is performed by reversing the two operands and making it a left outer join. When doing a left outer join, the cross join algorithm sets all inner data stream values to NULL when no key matching row is found in the inner data stream. It returns the nonmatched row from the outer data stream along with NULL values as part of the left outer join result.

5.6.2 Match Join

The match strategy has two variations:

- The general match variation, which uses either a sort with a temporary table or index retrieval
- A special match variation called the zigzag match, which does extra optimization when using index retrieval by employing index key skip on an incoming data stream

This section describes the general match strategy; Section 5.6.3 describes the zigzag match join.

When the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as S, the following lines indicate the general match retrieval strategy:

```
Match
  Outer Loop
     .
     .
     .
  Inner Loop
     .
     .
     .
```

The match strategy is more specific than the cross strategy. It contains two data streams, an outer loop and an inner loop. In this type of join, both loops must be sorted, either explicitly or by using an index. Rows in each loop are processed only once. Match join works only when the rows in the inner and outer loops are sorted in the same ascending order. In other words, descending indexes, if any, are not used by the match join method. In general, a match join is more efficient than a cross join because the match join scans each of the two loops once, while the cross join scans entry 2 (and higher entries) multiple times.

At the start of processing, the optimizer reads one row in both the inner and outer loops. If the join key values in the two rows are different, the optimizer keeps reading rows in the loop with the lower key value until the keys are equal in both loops. When a match of equal key values occurs, the optimizer checks for repetition of this matched key in both loops, and performs a cross join for this equal key group exactly as described in Section 5.6.1. The optimizer then repeats the process of finding matching keys.

For example, consider two tables, TABLE1 and TABLE2. TABLE1 has five rows, which return the values 1, 5, 5, 6, and 9. TABLE2 has six rows, which return the values 2, 3, 4, 5, 5, and 7. The match would process the two tables as shown in Example 5–1.

Example 5–1 General Match Strategy Processing

```
Outer Loop (TABLE1)             Inner Loop (TABLE2)

   **** Start search for a matching group ****

Read first key (1)               Read first key (2)

                      1 < 2

        Switch to Outer Loop to use lower key value

   ************ Skip unmatched keys ************

Read next key (5)
 5 > 2 (no match)
 Switch to Inner Loop
                                 Read next key (3)
                                  5 > 3 (no match)
                                 Read next key (4)
                                  5 > 4 (no match)
                                 Read next key (5)
                                  5 = 5 (match)

   ********* Process equal key group *********
        Perform the nested loop algorithm
             as in Cross strategy
        Deliver pairs of matching records

Use last-read 5                  Use last-read 5
            Deliver pair of records

Use last-read 5                  Read next key (5)
            Deliver pair of records
                                 Read next key (7)
Read next key (5)
(still the same equal key group)
```

(continued on next page)

Example 5–1 (Cont.) General Match Strategy Processing

```
                                 Restart at first key 5
Use last-read 5                  Use restarted 5
            Deliver pair of records

Use last-read 5                  Read next key (5)
            Deliver pair of records

                                  Read next key (7)
Read next key (6)
(equal key group has ended)

   **** Start search for a matching group ****

Use last-read 6                   Use last-read 7
                     6 < 7
        Switch to Outer Loop to use lower key value

   ************ Skip unmatched keys ***********

Read next key (9)
 9 > 7 (no match)
 Switch to Inner Loop
                                  Read EndOfData mark
                                  Finish matching
```

The optimizer tries to place the data stream with unique join keys in the outer loop. This eliminates backup of the inner loop data stream to the beginning of the equal key group.

In addition to an inner join, the match join algorithm can also perform a left outer join and full outer join. The right outer join is performed by reversing the two operands and making it a left outer join. When doing a left outer join, the match join algorithm sets all inner data stream values to NULL when no key matching row is found in the inner data stream. It returns the nonmatched row from the outer data stream along with NULL values as part of the left outer join result.

When doing the full outer join, the match join algorithm also sets all outer data stream values to NULL when no key matching row is found in the outer data stream. It returns the nonmatched row from the inner data stream along with NULL values as part of the full outer join result.

5.6.3 Match, Zigzag

The zigzag join strategy is a faster variation of the general match method. With this method, the data stream in the inner loop, outer loop, or both loops must be a table, and index retrieval must be available for that table. The zigzag strategy can be applied to the inner or outer leg.

When the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as S, the following lines indicate the zigzag match retrieval strategy:

```
Match
  Outer Loop  (zig-zag)
     .
     .
     .
  Inner Loop  (zig-zag)
     .
     .
     .
```

Zigzag processing differs from the regular match strategy described in Section 5.6.2. With the zigzag match variation, the optimizer can skip multiple keys in the inner and outer loops. The regular match strategy reads the inner loop keys one by one, comparing them with the outer loop key, and rejects the inner loop keys that do not match, one by one.

The zigzag match can skip over multiple keys in the inner loop by using the current outer loop key as a fresh point to restart index scan. The zigzag match can also skip over multiple keys in the outer loop by using the current inner loop key as a fresh point to restart index scan. When a large number of keys can be skipped, the restart scan procedure chooses to quickly descend from the top of the index B-tree to the fresh point, thus saving I/O operations and CPU time. When only a few keys need to be skipped, the restart procedure will not descend the B-tree but will perform an efficient in-memory skip saving CPU time.

Example 5–2 uses the same tables found in Example 5–1, and it shows a zigzag skip that occurs only on the inner loop. The outer loop is sorted and scanned. The inner loop has an index, which is used to access the key values.

Example 5–2 Zigzag Match Strategy Processing

```
Outer Loop (TABLE1)             Inner Loop (TABLE2)
                                Zigzag side

   **** Start search for a matching group ****

Read first key (1)               Read first key (2)

                      1 < 2

        Switch to Outer Loop to use lower key value

   ************ Skip unmatched keys ************

Read next key (5)
 5 > 2 (no match)
 Switch to Inner Loop
                                 Skip to next key >=5
                                 Skips keys 3,4
                                  Gets key 5
                                  5 = 5 (match)

   ********* Process equal key group *********

   Processing continues as in a regular match strategy
```

Oracle Rdb also supports zigzag key skip on the outer loop if the data stream in the outer loop is a table and index retrieval is used.

The zigzag match join's advantage over a regular match join is the ability to skip ahead to a higher key using the index on the inner, outer, or both loops.

5.6.4 Merge

The merge strategy concatenates data from two or more data sources. This strategy is invoked when the optimizer encounters the UNION operator.

When the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as S, the following lines indicate the merge retrieval strategy:

```
Merge block of n entries
   Merge block entry 1
       . . .
   Merge block entry 2
       . . .
   .
   .
   .
```

Unlike join operations, the merge strategy can return data common to several data sources as well as data contained in only one of the sources. In a merge strategy, two or more data sources are combined using an *append* function rather than a *match* function.

The merge strategy follows these general steps:

1. Entry 1 is processed completely according to the strategies described in the body of the entry.
2. Entry 2 is processed completely according to the strategies described in the body of the entry, and the result appended to the work from entry 1.

5.6.5 Join Ordering

One of the major functions of the query optimizer is to consider all possible ways in which tables can be joined together in order to identify the most efficient join order. However, the use of the following limit the range of possible solutions the optimizer will consider:

- Outer joins
- Derived tables
- Views

Contrary to inner joins and full outer joins, the left and right outer joins are noncommutative operations. That is, the operands of a left or right outer join cannot be reversed. For example, the join expression `A left outer join B` gives a different result than `B left outer join A`.

Also, contrary to inner joins, outer joins (left, right, or full) are nonassociative operations. This means the operands of an outer join cannot be permuted (or interchanged). For example, the join expression `(A inner join B) inner join C` gives the same result as `A inner join (B inner join C)`. Whereas, the join expression `(A full outer join B) full outer join C` gives a different result than `A full outer join (B full outer join C)`.

The following sections describe join ordering in further detail. Note that when not specified, "outer join" refers to a left or right or full outer join. Also note that in the following examples, the symbols IJ, LJ, RJ, and FJ are used to denote inner join, left outer join, right outer join, and full outer join, respectively.

5.6.5.1 Ordering of Join Operators

In a series of outer joins, SQL defines the default join ordering to be from left to right. However, you can override the default by using parentheses. For example, consider the join expression `A LJ B LJ C`.

Without parentheses, SQL will default to the join order `[A LJ B] LJ C` and require that you supply the ON or USING clauses as follows:

```
   A LJ B ON .... LJ C ON ....
```

You can use parentheses to alter the join ordering as follows:

```
   A LJ (B LJ C ON ....) ON .....
```

Note that the second expression does not produce the same results as the first, because the two orders are semantically different. Note also that the ON or USING clauses are not necessary if you use the NATURAL qualifier, effectively telling SQL to use the identically named columns in both tables for joining.

5.6.5.2 Ordering of Join Operands

The operands of a left or right outer join are noncommutative. That is, the operands cannot be reversed. For example, `A LJ B` is not the same as `B LJ A`. However, `A LJ B` produces the same results as `B RJ A`. This property is used by the optimizer to convert all right outer joins into left outer joins while still guaranteeing correct results. This conversion is done so that the join algorithm will have to perform only inner, left outer, and full outer joins.

Because inner joins and full outer joins are commutative, their operands can be reversed. The optimizer tries solutions by commuting their operands, and picking the solution that has the lowest cost.

5.6.5.3 Combining Join Operators

For semantically correct processing of outer joins, the operands of an outer join must not be permuted. Therefore, the outer join operands cannot be interchanged with operands of other joins.

For example, consider the join expression:

```
A IJ B ON ... IJ C ON ....
```

With inner joins, the optimizer is free to join A, B, and C in any order. It can first join A and C even though no explicit join condition exists between A and C. Doing so produces correct results and may provide the best performance.

By contrast, consider the following join expression:

```
A LJ B ON ... LJ C ON ....
```

This problem can be solved only by first joining operands A and B and then joining operand C.

In a mix of inner and outer joins, the outer joins enforce some restrictions on the join order. The two operands to each outer join are always joined together, and they do not permute with any other operands.

For example, consider the join expression:

```
A LJ B IJ C
```

There are only two possible join orders:

```
[A LJ B] IJ C
C IJ [A LJ B]
```

If LJ is replaced with FJ in the above join expression as follows:

```
A FJ B IJ C
```

Then there are four possible join orders:

```
[A FJ B] IJ C
[B FJ A] IJ C
C IJ [A FJ B]
C IJ [B FJ A]
```

In this example, the operands to the inner join as well as the operands to the full outer join can commute.

5.6.5.4 Changing the Join Order

It is not necessary to use the derived table construct to alter the join ordering unless you also want aggregation. Therefore, it is better to just use the parentheses to alter the join ordering, as in the following join expression:

```
A LJ (B LJ C ON ....) ON ....
```

The previous example is a better construct than the following join expression:

```
A LJ (B LJ C ON ....) as DTAB ON ....
```

The latter join expression can prevent the optimizer from generating an efficient strategy.

Note that the use of parentheses does not impose any restriction on the join ordering unless outer join semantics dictate such a restriction. For example, the following three join expressions are equivalent:

```
(A IJ B) IJ C
```

```
A IJ (B IJ C)
```

```
A IJ B IJ C
```

However, the following two join expressions are not equivalent:

```
(A IJ B) LJ C
A IJ (B LJ C)
```

In this case, the join order restriction is imposed by the presence of left outer join rather than the use of parentheses.

The use of derived tables and views imposes restrictions on the join ordering. For example, the following join expression forces the optimizer to always join A and B together; therefore, operands A and B are not allowed to permute with operand C.

```
(A IJ B) AS DTAB IJ C
```

The following query containing a view also forces the optimizer to always join A and B together:

```
CREATE VIEW V AS SELECT * FROM A NATURAL JOIN B;

SELECT * FROM V NATURAL JOIN C;
```

5.6.5.5 Using Derived Tables

SQL allows writing queries that include subqueries in various areas: the select list, the WHERE clause, and the FROM clause. Those subqueries can be nested.

A subquery in the FROM clause is called a **derived table**. An example of a derived table follows:

```
SELECT ... FROM
  (SELECT .... FROM .... WHERE ...) AS T1,
  (SELECT .... FROM
       (SELECT .... FROM .... WHERE ...) AS T3,
       (SELECT .... FROM .... WHERE ...) AS T4
   WHERE T3.x = T4.x ) AS T2
WHERE T1.y = T2.y;
```

Derived tables allow the easy expression of complex operations such as double aggregations or joining aggregations. However, using derived tables to perform only selections and joins, without any aggregations, can result in poor performance by limiting the choices available to the optimizer.

There is no reason to use derived tables if the query does not include aggregations. It is more efficient to “roll” the subqueries back into the parent query, a process called **query flattening**.

For example, the following query is inefficient:

```
SELECT E.EMPLOYEE_ID, E.LAST_NAME, J.JOB_CODE
 FROM
  (SELECT EMPLOYEE_ID, LAST_NAME
     FROM EMPLOYEES
    WHERE STATE = 'MA') E,
  (SELECT EMPLOYEE_ID, JOB_CODE
     FROM JOB_HISTORY
    WHERE JOB_END IS NULL) J
WHERE E.EMPLOYEE_ID = J.EMPLOYEE_ID;
```

The query is better expressed in the following way:

```
SELECT E.EMPLOYEE_ID, E.LAST_NAME, J.JOB_CODE
  FROM EMPLOYEES E, JOB_HISTORY J
 WHERE E.EMPLOYEE_ID = J.EMPLOYEE_ID
   AND J.JOB_END IS NULL
   AND E.STATE = 'MA';
```

When processing the first query, the optimizer considers each derived table as a separate entity and derives a query subplan for each of them. It then combines the results of the two subplans to obtain the final result. This is much less efficient than the second query, where the optimizer is able to produce one globally optimized query plan.

5.7 Dynamic Optimization

In the current version of Oracle Rdb, dynamic optimization is used only for a single table retrieval. In other words, it is a leaf-level dynamic optimization. During dynamic optimization, different table access strategies are run simultaneously, and each strategy produces some rows that satisfy the query. The strategy that seems to be working the fastest after a certain time period is selected to deliver the bulk of the rows. Static optimization, on the other hand, selects a table access strategy during query compilation time, leaving the execution time adjustments to dynamic optimization.

The static optimizer unavoidably makes mistakes in choosing the best retrieval strategy. One possible mistake is making the wrong choice between sequential and index retrieval; a second possible mistake is selecting a long index scan when a much shorter index range is available. Dynamic optimization improves upon traditional optimization techniques by using the correct index or combination of indexes.

Any number of tables can be optimized dynamically, but joins and unions are still translated by the static optimizer into an execution tree, with one table for each branch. Individual tables are accessed using one of the traditional strategies described in Section 5.5 or by one of the dynamic leaf-level strategies described in Section 5.7.2.

The query optimizer uses two aspects of dynamic optimization:

- Dynamic OR optimization
- Dynamic leaf-level optimization

5.7.1 Dynamic OR Optimization

The optimizer selects the dynamic OR optimization strategy when the following conditions exist:

- A sorted or hashed index is available.
- The query contains one of the following operators that define two or more index ranges:
  - IN
  - OR

Dynamic OR optimization offers the following advantages over traditional OR index retrieval (described in Section 5.5):

- It uses a single pair of NDX and GET blocks instead of one pair for each range. The index is opened and closed once instead of doing an index open and close for each range.
- It sorts the index ranges (either at query compilation time or at query execution time if range predicates involve host variables) on the low end of the key ranges; then, overlapping ranges are combined to eliminate scanning of redundant keys.
- It delivers results in index key order. This type of optimization eliminates the use of a sort, which is required if an explicit order is specified by the query, if the grouping of data for aggregation or duplicate elimination is specified by the query, or if query retrieval is processed as part of a match join. The following cases illustrate where dynamic OR optimization avoids the use of a sort to return results in index key order:
  - The following query illustrates the use of dynamic OR optimization, which helps avoid the use of a sort for a query that requests rows returned in an explicit order:

```
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME
   FROM EMPLOYEES
   WHERE EMPLOYEE_ID IN (’00345’, ’00166’, ’00222’)
   ORDER BY EMPLOYEE_ID;

Conjunct        Get     Retrieval by index of relation EMPLOYEES
  Index name  EMPLOYEES_HASH [1:1...]3

 EMPLOYEE_ID   FIRST_NAME   LAST_NAME
 00166         Rick         Dietrich
 00222         Norman       Lasch
 00345         James        Stornelli
3 rows selected
```

With traditional static OR optimization, the result would not have been delivered in index key order, and therefore, a sort would have been used to deliver the query result in the EMPLOYEE_ID order requested.

– The following query illustrates the use of dynamic OR optimization, which helps avoid the use of a sort when a query requires the grouping of data to compute a value with an aggregate built-in function:

```
SELECT E.EMPLOYEE_ID, AVG(SALARY_AMOUNT) AS AVG_SALARY
   FROM EMPLOYEES E, SALARY_HISTORY SH
   WHERE E.EMPLOYEE_ID = SH.EMPLOYEE_ID AND
         E.EMPLOYEE_ID IN (’00345’, ’00166’, ’00222’)
   GROUP BY E.EMPLOYEE_ID;

Aggregate       Conjunct
Match
  Outer loop
    Conjunct        Index only retrieval of relation EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [1:1...]3
  Inner loop      (zig-zag)
    Conjunct        Get     Retrieval by index of relation SALARY_HISTORY
      Index name  SH_EMPLOYEE_ID [1:1...]3

 E.EMPLOYEE_ID              AVG_SALARY
 00166            1.691766666666667E+004
 00222            1.307625000000000E+004
 00345            5.160700000000000E+004
3 rows selected
```

To illustrate dynamic OR optimization on sorted indexes, assume that a sorted index, DEG_COLLEGE_CODE, is defined on the COLLEGE_CODE column of the DEGREES table. The next example shows a query using the IN operator, which is shorthand notation for (COLLEGE_CODE =’USCA’ OR COLLEGE_CODE = ’FLU’). Following the query, the resulting strategy output is shown with the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter defined as S.

```
SQL> SELECT * FROM DEGREES
cont>   WHERE COLLEGE_CODE IN ('USCA', 'FLU');
Leaf#01 FFirst DEGREES Card=165
  BgrNdx1 DEG_EMP_ID [0:0] Fan=17
  BgrNdx2 DEG_COLLEGE_CODE [1:1...]2 Fan=17
 EMPLOYEE_ID   COLLEGE_CODE   YEAR_GIVEN   DEGREE   DEGREE_FIELD
 00187         FLU                  1966   BA       Arts
 00206         FLU                  1979   BA       Arts
 00231         FLU                  1967   BA       Arts
 00249         FLU                  1977   BA       Arts
 00415         FLU                  1976   MA       Applied Math
 00176         USCA                 1982   MA       Applied Math
 00198         USCA                 1974   BA       Arts
 00217         USCA                 1974   BA       Arts
8 rows selected
SQL>
```

The notation in the example is described in detail in Appendix C but the third line of output that begins “BgrNdx2” can be described as follows:

- DEG_COLLEGE_CODE indicates the index used for data retrieval.
- [1:1 . . .]2 indicates the optimizer is using dynamic OR index retrieval.
  - The first “1” indicates a low value for the segment in the first range (COLLEGE_CODE = 'USCA').
  - The second “1” indicates a high value for the segment in the first range (COLLEGE_CODE = 'USCA').
  - The “ . . . ” notation indicates other index ranges (in this case, one more range, COLLEGE_CODE = 'FLU').
  - The final “2” specifies the total number of index ranges before any overlapping ranges are combined.

When the optimizer executes the previous query, the following steps are performed:

1. The DEG_COLLEGE_CODE index is opened and a B-tree descent is performed, positioning at the low key of the first index range.
2. A get-next operation is performed to fetch a key from the index.
3. A Boolean check (COLLEGE_CODE = 'USCA' if the first range, and COLLEGE_CODE = 'FLU' if the second range) is done to determine if the key belongs to the current range.

4. If so, then a row is fetched and delivered, and steps 2 to 4 are repeated until Boolean check at step 3 fails. Otherwise, a key-skip is performed using the low key of the next range (if any) to gain a start position at the beginning of the next range and steps 2 to 4 are repeated to scan keys from the new range.
5. When all index ranges are scanned, the index is closed.

The key-skip mechanism is optimized so that, depending on the size of the gap between the ranges, a B-tree descent may or may not be performed to locate the beginning of the next range.

Consider another example using a two-segment index. Assume a sorted index DEG_COLLEGE_CODE_YEAR_GIVEN exists on the COLLEGE_CODE and YEAR_GIVEN columns of the DEGREES table. Dynamic OR optimization is used for the following query:

```
SQL> SELECT * FROM DEGREES
cont> WHERE COLLEGE_CODE = ’STAN’ AND
cont>       (YEAR_GIVEN = 1973 OR YEAR_GIVEN = 1981);
Leaf#01 FFirst DEGREES Card=165
  BgrNdx1 DEG_COLLEGE_CODE_YEAR_GIVEN [2:2...]2 Fan=17

EMPLOYEE_ID   COLLEGE_CODE   YEAR_GIVEN   DEGREE   DEGREE_FIELD
00185         STAN           1973         MA       Elect. Engrg.
00201         STAN           1973         BA       Arts
00208         STAN           1981         BA       Arts
00226         STAN           1981         BA       Arts
00230         STAN           1981         MA       Statistics
00374         STAN           1981         PhD      Statistics
6 rows selected
```

Two index ranges are specified in the query:

- The first range is COLLEGE_CODE = ’STAN’ and YEAR_GIVEN = 1973
- The second range is COLLEGE_CODE = ’STAN’ and YEAR_GIVEN = 1981

The notation [2:2 . . .]2 indicates that the optimizer uses dynamic OR index retrieval, that there are low values for two segments and high values for two segments in the first range, and there are a total of two index ranges.

5.7.2 Dynamic Leaf Optimization

Dynamic leaf optimization (sometimes referred to as DynamOpt) chooses the best combination of indexes when several useful indexes are available. An index scan yields a highly accurate count of data that will result from the query. The selectivity estimates resulting from a scan are used to sequence the available indexes in order of scan cost to determine the best index or indexes to use.

The optimizer chooses the dynamic leaf strategy automatically when at least one index with a range or equality restriction is available, and the query does not perform any updates on the table. In four cases the optimizer selects a traditional index retrieval strategy to avoid dynamic leaf optimization's experimental overhead. The four exceptions include:

- A unique index with exact key match (that is, all segments are specified as equalities)
- An index only retrieval where no other useful indexes exist
- An index only retrieval that delivers rows in requested order (other useful indexes can exist but will be ignored)
- An OR index retrieval that uses two or more indexes to access data from a table

Dynamic leaf strategies cover most major requirements and index configurations for single table retrieval. The four dynamic leaf-level retrieval strategies include:

- Background Only (shown as "BgrOnly" in the RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS strategy display), which optimizes total retrieval time. This strategy is described in Section 5.7.2.1.
- Fast First ("FFirst"), which optimizes retrieval time for the first few rows of a query. This strategy is described in Section 5.7.2.2.
- Index Only ("NdxOnly"), which the optimizer uses when an index contains all the columns required by a query, and one or more other useful indexes exist that require a data fetch. This strategy is described in Section 5.7.2.3.
- Sorted ("Sorted"), which the optimizer uses when an existing index provides the requested sorted order, and one or more useful indexes exist. This strategy is described in Section 5.7.2.4.

All four strategies are governed by two major principles:

- Strategies compete against each other; the winner is selected to deliver the rows.

- During competition, useful data is accumulated from each index scan to help other scans and fetches to be more effective.

The primary benefit of dynamic leaf-level optimization is the provision of near-optimal performance for each instance of table access, even within the same query run, regardless of:

- Data distribution
- Columns correlation
- Whether zero or small numbers of selected rows are delivered in one instance
- When all or many rows are delivered in the other instance of the same leaf execution

There are two dynamic optimization contexts: the *background* process and the *foreground* process. The two processes are implemented within a single thread of either a user program or within interactive SQL. Pieces of individual scans run interchangeably, maintaining the same proportional speed of advancement over the rows.

The Background Process

The background process scans ranges of one or more indexes and delivers a sorted list of dbkeys and a filter represented either as an in-memory sorted dbkey list or an in-memory bitmap. Filters are used by the foreground process to avoid unnecessary row fetches from the data areas. The sorted dbkey list is used by the Fin stage for the final data row fetches and delivery. The presorting of dbkeys for delivery improves the efficiency of queries that fetch data that is not placed using an index. The background process cannot deliver rows to the user.

The background process scans each background index (noted as BgrNdx1, BgrNdx2, . . . BgrNdxN) and stores dbkey lists in memory buffers or in temporary tables. Each dbkey list is usually smaller than the previous list because dbkeys that are not in the previous dbkey list are discarded. As a result, only the current and last completed lists of dbkeys are maintained at any one time.

When the static optimizer selects two or more background indexes, the initial index sequence is based on rough estimates of the number of dbkeys that satisfy each index restriction. However, the actual number of dbkeys can change at each new leaf node invocation because the values of the variables involved in restrictions can change.

To provide better dbkey number estimates for each leaf invocation, dynamic optimization runs a quick estimate procedure for each background index at each leaf invocation. The purpose of this estimation is to change the sequence of the background indexes to ascending selectivity order. The sequence may change from one leaf invocation to another because range-defining variables can change. By resequencing based on fresh variable values, the optimality of index usage is assured even if range sizes go up and down with each leaf invocation. In some cases, dynamic estimates are not performed if the estimation procedure would take an excessive amount of time, such as when:

- The range list containing the logical OR is involved in a restriction, as described in Section 5.7.1
- An index is stored in more than one storage area

Table 5–1 shows the conditions that can cause background index scan termination. The table also shows the execution trace output (RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS defined as E) that indicates each condition.

Table 5–1 Conditions That Cause Background Index Scan Termination

Condition	E Flag Output
Whole index range scanned to completion.	EofData
Too many dbkeys are read (threshold limit reached). The background process scans other available indexes.	ThreLim
Too many I/O operations done (fetch limit reached). The background index tries the next index.	FtchLim
Explicit command issued, such as “Close Leaf.”	Termin*

Note

Table C–3 lists and defines all output notation for the RDMS$DEBUG_FLAGS and RDB_DEBUG_FLAGS E flag.

The background process never delivers rows to a caller (hence the term background). When the background process terminates, dynamic leaf-level optimization usually switches to the final (Fin) stage. The Fin stage delivers data rows based on:

- The dbkey list collected in a memory buffer. This is noted as “Fin Buf” in the strategy output.
- The dbkey list collected in a temporary table. This is noted as “Fin TTbl”.

The Fin stage never switches to another stage, and is never interrupted except by the explicit command “Close Leaf (- ‘CUT)”

The Foreground Process

The second component of dynamic leaf-level optimization is the foreground process, which either scans a foreground index or borrows dbkeys from the background process. The foreground process runs in parallel with the background process. The foreground process fetches and delivers rows immediately after reading a dbkey from an index or receiving the dbkey from the background process.

Section 5.7.2.1 through Section 5.7.2.4 describes the four leaf types used by dynamic optimization to retrieve data rows.

5.7.2.1 Background Only Retrieval

The most important component of dynamic optimization is the background process. This process scans one or more indexes and delivers a sorted list of dbkeys in ascending order to the final (Fin) stage of retrieval. The Fin stage then delivers the data rows. The background only leaf strategy (BgrOnly) optimizes a query for total time retrieval—that is, BgrOnly finds all the rows that satisfy the query before it delivers them. The foreground process, which delivers rows immediately, is not run as part of the BgrOnly strategy.

The optimizer selects the background only leaf strategy when one or more indexes could be used for retrieval but none of them contain all the columns required by the query.

The background only leaf strategy executes the following generalized steps:

1. Scans all useful indexes prioritized by their estimated selectivities. The shortest index is scanned first.

2. Stores dbkeys in a dbkey list.

3. Abandons an index scan if the dbkey list length exceeds the length of a dbkey list already built. The decision to abandon an unproductive index is based on the actual length of the growing dbkey list, not on the error-prone selectivity estimation. Additionally, the optimizer counts the actual fetches (physical I/Os) for each index scan and limits experimentation to some proportion of the cost of the best retrieval method available at a given moment.

4. While scanning, weeds out dbkeys not contained in previously built dbkey lists.

5. Enables row fetches by the Fin stage using dbkeys from the shortest completed dbkey list.

6. Sorts the shortest complete dbkey list before handing it to the Fin stage in order to avoid duplicate reads of the same data pages.

The background process optimizes a total retrieval time by guaranteeing use of the best subset of available indexes. You do not have to be concerned about whether a query restriction covers a single row or 90 percent of a table's rows, or whether one or more indexes exist for the restriction. However, in a low update environment, defining several few-segment indexes can be more efficient than defining a single multisegment index.

Oracle Rdb allows you to specify a total time retrieval strategy in your query. See Section 5.8.5 for more information.

5.7.2.2 Fast First Retrieval

In contrast to the total time retrieval strategy of the BgrOnly leaf, some queries or applications need only look at the first or first several rows returned by a query. For example, the EXISTS predicate only checks for the existence of a single data row. Also, an interactive user typically looks at the first screen or the first several screens of query results, not thousands of them, before canceling the query. In this case, the optimizer uses the fast first (FFirst) retrieval leaf strategy.

Fast first retrieval is used when the following conditions are true:

- At least one applicable index exists
- No index exists that enables index only retrieval
- No sort is needed

Because the background process cannot directly deliver rows, the optimizer runs the foreground process in parallel with the background process for FFirst retrieval. The foreground process can deliver rows immediately.

The FFirst leaf strategy executes the following generalized steps:

1. The background process opens the best index and begins scanning. The background process passes each dbkey from the index scan to the foreground process.
2. The foreground process begins delivering rows immediately and stores the dbkeys of delivered rows in a foreground buffer. Sometimes when the fast first strategy is selected for record retrieval, no records or very few records are selected. In this case, the foreground process of the leaf node does many record fetches in an attempt to deliver them, but they all (or almost all) are rejected based on the selection criterion.

3. If an explicit Close Leaf command is received by the leaf while background and foreground activities are in progress, it indicates the user (or the user's program) or other parts of a query that called this leaf were satisfied with the few rows already delivered by the foreground process and needed no more rows. The fast first retrieval strategy employs the foreground process precisely because of a high expectation of such a premature retrieval termination. Upon premature termination, both background and foreground processes stop and leaf controlled resources, including the foreground buffer, are released.
4. A competition occurs between the foreground and background processes in which foreground fetches are terminated and a switch is made to the background only (BgrOnly) strategy when the cost of the foreground fetches exceeds half of the projected background cost. This feature reduces unnecessary foreground cost and chooses the guaranteed efficient strategy as soon as the chance of success by the foreground process becomes unreasonably low. This competition mechanism provides better performance when the user has incorrectly chosen fast first optimization or when the previously unknown data distribution makes the total time optimization more efficient than the fast first optimization.

   When the optimizer terminates the foreground process, it does not throw away the foreground buffer. It still needs to know which rows have been delivered so that the same row does not get delivered twice. The background continues to obtain rows and pass the dbkeys to the Fin stage. The Fin stage filters the dbkeys through the foreground buffer and delivers the remaining rows.
5. If the background process finishes before an explicit Close Leaf command, the foreground terminates. The background process switches to the Fin stage and delivers the rows.

Oracle Rdb allows you to specify a fast first retrieval strategy in your query. See Section 5.8.5 for more information.

5.7.2.3 Index Only Retrieval

The optimizer uses the index only (NdxOnly) leaf retrieval strategy when an index exists that contains all the columns needed to satisfy a query. At least one other index must also be available or the optimizer will use traditional index only retrieval.

The index only leaf strategy executes the following generalized steps:

1. The optimizer starts both background and foreground processes. The foreground process opens the index that contains all the columns necessary to complete the query. The background process opens an index from the

remaining available indexes that it estimates will yield the fewest dbkeys. Upon opening, both processes advance with proportional speed of incurred cost, measured in physical I/O operations.

2. The foreground process starts delivering rows immediately. When a row is delivered, the optimizer records the row's dbkey in a buffer. If the buffer overflows, the background process ends and the index only scan continues until completion. If the buffer does not overflow, when the background dbkey list becomes available, it is used for the final row fetches (the NdxOnly scan terminates at this point).

Consider what happens in the background process. The background slowly builds a dbkey list for the best index or indexes by scanning them, the best candidates first, and keeping no more than two indexes open at the same time. If background processing completes its dbkey list before the foreground buffer overflows, then the foreground process is terminated. Because the background process cannot deliver rows, the optimizer moves to the Fin stage. To avoid outputting the same data twice, the dbkeys from the background dbkey list are checked against the list of already-output dbkeys recorded in the foreground buffer. The rows that do not match those in the foreground buffer are delivered to the caller.

5.7.2.4 Sorted Order Retrieval

The optimizer uses the sorted order leaf retrieval strategy when the following conditions exist:

- The query specifies or implies a particular sort order, for example, when aggregates, projections, or match strategies demand a specific sort order.
- Index only retrieval is not possible.
- An index with the correct order exists.
- At least one other index exists that can be used for dbkey filtering.

The sorted order leaf strategy executes the following generalized steps:

1. The optimizer starts both background and foreground processes. The foreground process opens the sorted index and begins to deliver rows immediately. The dbkeys of the delivered rows are not saved in a buffer. The background process begins scanning other useful indexes in parallel with the foreground process. The background does not race with the foreground process; instead, it builds a dbkey list or bitmaps for filtering purposes.
2. When the background process finishes, it releases all resources and retains only the final dbkey list or the final bitmap filter.

3. The rows that the foreground process continues to deliver are now checked (filtered) against the background dbkey list or bitmap. If the dbkey matches, the row is retrieved from the data page. Otherwise, the row is not fetched, thus saving I/O.

5.8 Working with the Query Optimizer

This section describes ways to maximize the optimizer's potential.

5.8.1 Views and Query Optimization

Views can assist the optimizer in developing the best retrieval strategy if the view is well-defined.

Views present an opportunity to simplify the optimization task and thus reduce the total time spent in determining a retrieval strategy. If a view of three tables is joined with a fourth table in a query, the optimizer will derive a retrieval strategy for the view and then derive a retrieval strategy for the query (that is, a view join table). The view is treated as a single table whose cardinality is the estimate of the resulting data stream of the view. In other words, the view retrieval strategy is determined separately from the rest of the query. To produce an efficient view retrieval strategy, any predicates that are based on view columns but present outside of a view are also used. Similarly, the cardinality estimate of the view is used when retrieval strategy for the query is produced. Thus a view is optimized separately from the query, but global information is used when appropriate.

Consider the following example. Four tables need to be joined: TAB_A, TAB_B, TAB_C, and TAB_D. If TAB_A, TAB_B, and TAB_C are combined in a view, the number of possible solutions that the optimizer must consider (assuming no indexes and equal cardinality) is 3! plus 2! (or $(3 * 2 * 1 + 2 * 1 = 8)$) as opposed to 4! (or $(4 * 3 * 2 * 1 = 24)$) if no view is used and all four tables are of equal cardinality and no indexes are joined. However, the degree to which each solution is considered will be different, because solution pruning will take place. Because the number of possible solutions is reduced when a view is used, the total time to optimize the query is also reduced.

Consider another example. There are four tables: TAB_A, TAB_B, TAB_C, and TAB_D. If TAB_A and TAB_B are combined in VIEW_1, and TAB_C and TAB_D are combined in VIEW_2 with no join between the views, the optimizer derives a retrieval strategy for VIEW_1 (first join), a retrieval strategy for VIEW_2 (second join), and then a retrieval strategy for cross-product of the results from VIEW_1 and VIEW_2. But if all four tables are used in a query, then the optimizer produces a retrieval strategy with a join of two tables, followed by a cross-product with third table, and finally a join with fourth table. Because cross-product is not the last operation performed in the second case, it is likely that the performance of such a retrieval strategy is worse than the retrieval strategy produced for a query with views.

5.8.2 Concatenated Expressions and Query Optimization

Queries that include comparisons between concatenated expressions may sometimes be decomposed into equivalent expressions that enable the use of an index or indexes. The following are two possible cases:

```
SELECT * from table where a || b = c || d;             (case A)
SELECT * from table where a || b > 'Some string';      (case B)
```

In each of these cases, if a is a fixed-width text column and an index is defined on a, that index should be usable. In case A, the width of column a must be identical to the width of column c.

If one or more indexes include a or b as the leading segment of the index, you can use a fast index lookup. A fast index lookup is much faster than the performance of a full index scan.

The following restrictions apply to the optimization of concatenated expressions:

- The width of a and :aVar must be identical. (This restriction applies to case A.)
- A usable index must be defined on column a. (This restriction applies to case A.)
- Column a must be a fixed-width text column. (This restriction applies to cases A and B.)

5.8.3 Queries Not Directly Based on the First Key Segment

A situation to avoid (or at least know the consequences of) is a query with the following characteristics:

- Directly or indirectly uses OR logic
- Uses an index with segmented keys
- Does not directly depend on the first key segment

Queries that use AND logic always perform better than those that use OR logic because using AND results in a much smaller subset of dbkeys. OR logic concatenates OR lists and results in a list equal to the sum of two lists. Table 5–2 shows the results of AND and OR logic.

Table 5–2 AND and OR Logic Compared

Condition	Number of Queries	Dbkey Set Size
AND	1	Small
OR	2	Large, concatenated

For segmented keys, where two or more columns are used to uniquely identify a row, you must be careful how you represent the segments in a query. For example, consider the following three-part key named EMP_JOB_DEPT:

- The EMPLOYEE_ID column forms the first key segment.
- The JOB_CODE column forms the second key segment.
- The DEPARTMENT_CODE column forms the third key segment.

You should base a query on the first segment of the key and include the second and third segments using OR logic. The best approach is to avoid the detrimental OR condition. *If* you form a query that uses the first segment of the key and the AND condition with the remaining two segments, both of which have the OR condition, you can run into problems. To represent this mathematically:

```
Where, S1=EMPLOYEE_ID, S2=JOB_CODE, and S3=DEPARTMENT_CODE
S1 = X AND ((S2 = Y AND S3 = Z) OR (S2 =Y1 AND S3 = Z1))
```

By representing the query in this way, the statement between the outer parentheses, (S2 = Y AND S3 = Z) OR (S2 =Y1 AND S3 = Z1), is not directly based on segment 1. Instead, you have an OR condition between segments 2 and 3 that results in an indexed search based on segment 1. To be certain that segments 2 and 3 are directly based on segment 1, it is better to represent the query in the following way:

```
(S1 = X AND S2 = Y AND S3 = Z) OR (S1 =X AND S2= Y1 AND S3 = Z1)
```

To summarize, it is best to avoid using queries that are not directly based on the first key segment.

5.8.4 Index Placement

Dynamic optimization assumes that data rows are clustered by some sorted index only if the PLACEMENT VIA clause is specified in the CREATE STORAGE MAP statement for the table, or if another sorted index has one or more of its leading segments identical to the leading segments of the PLACEMENT VIA index. All other non-PLACEMENT VIA indexes are assumed to have no clustering effect; that is, random physical row fetches are expected from such index scans, yielding at least one I/O operation for each single fetched row. If you define a PLACEMENT VIA index and then drop it, or if some non-PLACEMENT VIA index has a strong statistical correlation to the PLACEMENT VIA index, or if the table just happens to be loaded in the same sequence as some index order, then such an implicit clustering effect is hidden from the optimizer.

To ensure better performance, you should specify a clustering order explicitly. However, even with implicit clustering, the dynamic optimizer detects and measures the clustering factor when more than one index scan of the same index is performed. This means that with multiple executions of the same compiled query or with multiple index retrievals within a single query run, the optimizer adapts to arbitrary clustering and can approach the best performance rate.

5.8.5 Specifying a Preferred Optimization Mode

Applications and 4GL tools can influence the retrieval strategy selected by the query optimizer when you specify either fast first row retrieval or total time retrieval.

- With fast first row retrieval, data is returned as soon as possible. This strategy benefits applications that need to establish the existence of, or look at, one or a few rows that satisfy the query. For example, interactive applications can allow a user to abort a query after displaying several screens of data without reading the entire set of rows. See Section 5.7.2.2 for more information on the fast first retrieval strategy.
- With total time retrieval, the optimizer determines the best strategy for satisfying the entire query. This strategy benefits applications, such as batch jobs, that are interested in minimizing total retrieval time and can afford to wait for the first row of data to be returned. Total time retrieval is always used by the background only retrieval strategy. See Section 5.7.2.1 for more information on the background only retrieval strategy.

Note that, despite the presence of a preferred option, Oracle Rdb can override a specific option when it detects a solid reason for doing so. For example, the EXISTS predicate dictates fast first optimization, whether or not the total time option was specified for the query. Aggregate expressions (AVG, COUNT, MAX, MIN, and SUM) with no GROUP BY clause dictate total time optimization, whether or not the fast first option was specified for the query.

SQL enables you to specify optimizer preferences using:

- The SELECT expression
- The singleton form of the SELECT statement
- The SQLOPTIONS qualifier on the precompiler command line
- The OPTIMIZATION_LEVEL qualifier on the module language command line
- The SET OPTIMIZATION LEVEL statement

The SELECT expression and singleton form of the SELECT statement are used to specify optimizer preferences for individual statements. The SQLOPTIONS=OPTIMIZATION_LEVEL qualifier on the precompiler command line is used to specify an optimizer preference for a majority of the statements in a precompiler program. The OPTIMIZATION_LEVEL qualifier on the module language command line is used to specify an optimizer preference for a majority of the statements in a module language program. The SET OPTIMIZATION LEVEL statement is used to specify an optimizer preference for a majority of the statements in dynamic SQL or interactive SQL.

The rest of this section provides more information on specifying optimizer preferences using SQL. Refer to the *Oracle Rdb7 SQL Reference Manual* for syntax details.

- The SELECT expression

  The OPTIMIZE FOR { DEFAULT | FAST FIRST | TOTAL TIME } clause can be specified with the SELECT expression to specify the optimization mode Oracle Rdb will use for the SELECT expression. Note that the SELECT expression can be used with the DECLARE CURSOR statement, with the INSERT statement, and with the interactive SELECT statement.

  The following interactive SQL example shows two DECLARE CURSOR statements that set the preferred optimization mode for a cursor. The first DECLARE CURSOR statement sets the optimization mode for fast first retrieval. The second DECLARE CURSOR statement sets the optimization mode for total time retrieval:

```
SQL> DECLARE CEMP TABLE CURSOR
cont>  FOR
cont>    SELECT    *
cont>      FROM   EMPLOYEES
cont>      WHERE  EMPLOYEE_ID > '01100'
cont>  OPTIMIZE FOR FAST FIRST;
SQL>
SQL> DECLARE TEMP TABLE CURSOR
cont>  FOR
cont>    SELECT    LAST_NAME, FIRST_NAME
cont>      FROM   EMPLOYEES
cont>  OPTIMIZE FOR TOTAL TIME;
```

The following interactive SQL example shows how you can choose an optimization strategy for a SELECT statement:

```
SQL> SELECT * FROM EMPLOYEES
cont> OPTIMIZE FOR FAST FIRST;
SQL>
SQL> SELECT * FROM EMPLOYEES
cont> OPTIMIZE FOR TOTAL TIME;
```

- The singleton form of the SELECT statement

  The OPTIMIZE FOR { DEFAULT | FAST FIRST | TOTAL TIME } clause can be specified with the singleton SELECT statement to specify the optimization mode Oracle Rdb will use for the singleton SELECT statement. Note that the singleton SELECT statement is valid only in SQL module language and SQL precompiler programs.

- SQLOPTIONS=(OPTIMIZATION_LEVEL=DEFAULT | FAST_FIRST | TOTAL_TIME)

  The SQLOPTIONS=(OPTIMIZATION_LEVEL=DEFAULT | FAST_FIRST | TOTAL_TIME) qualifier for the SQL precompiler command line specifies the default optimizer strategy to be used for processing those statements embedded in SQL precompiler programs that do not specify a particular optimizer strategy with the OPTIMIZE FOR clause.

  Use the SQLOPTIONS=(OPTIMIZATION_LEVEL=DEFAULT | FAST_FIRST | TOTAL_TIME) qualifier to select the optimizer strategy that is appropriate for the majority of the statements in your precompiler program. Individual statements in a precompiler program that use the OPTIMIZE FOR syntax to specify an optimizer strategy will be processed with the optimizer strategy specified with the OPTIMIZE FOR syntax.

- OPTIMIZATION_LEVEL=(DEFAULT | FAST_FIRST | TOTAL_TIME)

The OPTIMIZATION_LEVEL=(DEFAULT | FAST_FIRST | TOTAL_TIME) qualifier for the SQL module language command line specifies the default optimizer strategy to be used for processing those statements in SQL module language programs that do not specify a particular optimizer strategy with the OPTIMIZE FOR clause.

Use the OPTIMIZATION_LEVEL=(DEFAULT | FAST_FIRST | TOTAL_TIME) qualifier to select the optimizer strategy that is appropriate for the majority of the statements in your module language program. Individual statements in a module language program that use the OPTIMIZE FOR syntax to specify an optimizer strategy will be processed with the optimizer strategy specified with the OPTIMIZE FOR syntax.

- SET OPTIMIZATION LEVEL { 'DEFAULT' | 'FAST FIRST' | 'TOTAL TIME' }

  The default optimization mode for statements in dynamic SQL is the mode specified for the module in which the statement is prepared. The SET OPTIMIZATION LEVEL statement is used to override the default optimization mode that is set for the module.

  For interactive SQL, the default optimization mode is DEFAULT. The SQL SET OPTIMIZATION statement can be used in interactive SQL to override the default optimization mode.

  The following interactive SQL examples show how you can use the SET OPTIMIZATION LEVEL statement to request the fast first retrieval strategy or total time retrieval strategy for one or more queries.

  Use the following statement if the majority of the queries in your interactive session are *not* going to be closed before all the result records are delivered:

  ```
  SQL> SET OPTIMIZATION LEVEL 'TOTAL TIME';
  ```

  Use the following statement if the majority of the queries in your interactive session *are* going to be terminated after examining the first few records (that is, before all the result records are delivered):

  ```
  SQL> SET OPTIMIZATION LEVEL 'FAST FIRST';
  ```

  The following statement sets the optimization back to the default behavior (when no SET OPTIMIZATION LEVEL strategy has been specified). The default behavior is to try the fast first retrieval strategy first, then select the total time retrieval strategy if it will retrieve the records faster than the fast first strategy:

  ```
  SQL> SET OPTIMIZATION LEVEL 'DEFAULT';
  ```

If different portions of your interactive session need different optimization levels, use the SQL SET OPTIMIZATION LEVEL statements to set the desired level before each such portion.

Example 5–3 shows the use of the SET OPTIMIZATION LEVEL statement. The example assumes that you have defined the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter as S.

Example 5–3 Using the SQL SET OPTIMIZATION LEVEL Statement

```
$ SQL
SQL> ATTACH ’FILENAME personnel’;
SQL> --
SQL> -- No optimization level has been selected.  The optimizer
SQL> -- selects the fast first (FFirst) retrieval strategy to
SQL> -- retrieve the rows from the EMPLOYEES table in the
SQL> -- following query:
SQL> SELECT EMPLOYEE_ID, LAST_NAME
cont>  FROM EMPLOYEES
cont>  WHERE EMPLOYEE_ID IN (’00167’, ’00168’);
Leaf#01 FFirst RDB$RELATIONS Card=19
  BgrNdx1 RDB$REL_REL_NAME_NDX [1:1] Fan=8
Sort
Cross block of 2 entries
  Cross block entry 1
    Leaf#01 BgrOnly RDB$RELATION_FIELDS Card=71
      BgrNdx1 RDB$RFR_REL_NAME_FLD_ID_NDX [1:1] Fan=8
  Cross block entry 2
    Get     Retrieval by index of relation RDB$FIELDS
      Index name  RDB$FIELDS_NAME_NDX [1:1]  Direct lookup
Leaf#01 FFirst EMPLOYEES Card=100
  BgrNdx1 EMP_EMPLOYEE_ID [1:1...]2 Fan=17
 EMPLOYEE_ID   LAST_NAME
 00167         Kilpatrick
 00168         Nash
2 rows selected
SQL> --
SQL> -- Use the SET OPTIMIZATION LEVEL statement to specify that you want
SQL> -- the total time (BgrOnly) retrieval strategy to be used.  Note that
SQL> -- when the previous query is executed again, the total time (BgrOnly)
SQL> -- retrieval strategy is selected, instead of fast first:
SQL> SET OPTIMIZATION LEVEL ’TOTAL TIME’;
SQL> SELECT EMPLOYEE_ID, LAST_NAME
```

(continued on next page)

Example 5–3 (Cont.) Using the SQL SET OPTIMIZATION LEVEL Statement

```
cont>  FROM EMPLOYEES
cont>  WHERE EMPLOYEE_ID IN (’00167’, ’00168’);
Leaf#01 BgrOnly EMPLOYEES Card=100
  BgrNdx1 EMP_EMPLOYEE_ID [1:1...]2 Fan=17
 EMPLOYEE_ID   LAST_NAME
 00167         Kilpatrick
 00168         Nash
2 rows selected
SQL> --
SQL> -- When you specify the SET OPTIMIZATION LEVEL ’DEFAULT’ statement,
SQL> -- either the fast first or total time strategy will be selected.
SQL> -- The fast first strategy will be tried first, then total time
SQL> -- will be selected if it will retrieve the rows faster than the
SQL> -- fast first strategy.
SQL> SET OPTIMIZATION LEVEL ’DEFAULT’;
SQL> --
SQL> -- Because the fast first strategy is faster than the total
SQL> -- time strategy for this query, the fast first strategy
SQL> -- is used to retrieve the rows:
SQL> SELECT EMPLOYEE_ID, LAST_NAME
cont>  FROM EMPLOYEES
cont>  WHERE EMPLOYEE_ID IN (’00167’, ’00168’);
Leaf#01 FFirst EMPLOYEES Card=100
  BgrNdx1 EMP_EMPLOYEE_ID [1:1...]2 Fan=17
 EMPLOYEE_ID   LAST_NAME
 00167         Kilpatrick
 00168         Nash
2 rows selected
SQL>
```

If you use an initialization procedure (the sqlini.sql procedure that is automatically executed in the beginning of each session), you can set the most commonly used optimization level in the initialization procedure.

5.8.6 Using the Query Governor

In some database environments, interactive or application users can consume excessive system resources by entering general queries that require multiple table joins or that return every row in a table. By default, Oracle Rdb runs every query to completion. You can prevent the overload caused by lengthy queries by setting limits, within an application or environment, that restrict query output. The mechanism that restricts output is called the **query governor**. You can use the query governor to set one or more of the following limits:

- You can restrict output by limiting the number of rows a query can return. The optimizer counts each row returned by the query and stops execution when the row limit is reached. Note that the row limit value is independent of the number of rows read by the application, such as intermediate rows for a join or component rows for an aggregate. However, avoid setting too low a value because the row limit value applies to all database queries, including queries to system tables (for example, by SQL) that fetch the metadata required to parse a query. If the row limit value prevents access to the required metadata, the attempt to execute the query will fail.

- You can restrict output by limiting the amount of elapsed time the optimizer spends compiling a query. The time limit value is an integer that specifies the number of elapsed seconds.

Note

Specifying a query compilation time limit can cause application failure in certain circumstances. An application that runs successfully during off-peak hours when the system load is light may time out when it is run during peak hours.

- You can restrict the amount of CPU time used to optimize a query for execution. The CPU time limit value is an integer that specifies the number of CPU seconds. The default is unlimited CPU time for the query compilation.

In all three cases, users receive an error message if a limit is exceeded. For example, if a user exceeds the row limit, Oracle Rdb displays the following message:

```
%RDB-E-EXQUOTA, Oracle Rdb runtime quota exceeded
-RDMS-E-MAXRECLIM, query governor maximum limit of records has been reached
```

You can set the row limit and time limit values by using any of the following methods:

- For interactive SQL, use the SET QUERY LIMIT statement.

  ```
  SQL> SET QUERY LIMIT TIME h
  SQL> SET QUERY LIMIT ROWS i
  SQL> SET QUERY LIMIT CPU TIME j
  ```

  You can use the SHOW QUERY LIMIT statement to display the values set for each limit.

```
SQL> SHOW QUERY LIMIT
  QUERY LIMIT TIME limit is h seconds
  QUERY LIMIT ROWS limit is i rows
  QUERY LIMIT CPU TIME limit is j seconds
```

- For the SQL precompiler, use the query limit parameter to the SQLOPTIONS qualifier.

  ```
  /SQLOPTIONS = (QUERY_TIME_LIMIT = h, QUERY_MAX_ROWS = i,
                        QUERY_CPU_TIME_LIMIT = j)
  ```

- For the SQL module language processor, use the following qualifiers:

  ```
  /QUERY_TIME_LIMIT = h  /QUERY_MAX_ROWS = i  /QUERY_CPU_TIME_LIMIT = j
  ```

- For dynamic SQL, options are inherited from compilation qualifiers.
- Use the logical names RDMS$BIND_QG_TIMEOUT, RDMS$BIND_QG_REC_LIMIT, and RDMS$BIND_QG_CPU_TIMEOUT or the configuration parameters RDB_BIND_QG_TIMEOUT, RDB_BIND_QG_REC_LIMIT and RDB_BIND_QG_CPU_TIMEOUT. Refer to Section A.85, Section A.84, and Section A.83, respectively, for details.

OpenVMS VAX OpenVMS Alpha

If you are using RDO, the only way you can set query limits is to define these logical names. ♦

With all of the query governor options, you can set the time limit, the row limit, and the CPU time limit. Whichever value is reached first stops the output.

5.8.7 Using RDMS$DEBUG_FLAGS and RDB_DEBUG_FLAGS

You can use the logical name RDMS$DEBUG_FLAGS or the configuration parameter RDB_DEBUG_FLAGS to examine how the optimizer executes a query. By analyzing query statistics and query strategies, you may be able to improve performance. This section describes some things to look for when you use the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter. For a complete description and sample output, refer to Appendix C.

Sequential Retrieval

To improve performance of a query, pay particular attention to the notation, “Retrieval sequentially,” in the RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS strategy display (S flag). This notation indicates that the optimizer is sequentially searching an entire table. You may be able to improve performance by defining an index for the column or columns the query is based on. You should not assume, however, that simply defining indexes will automatically speed up the query, or that the optimizer will use the index you have defined. In some cases, the optimizer ignores the index because it is faster to access rows sequentially. Sequential access generally works best for small tables that are stable and contain a small number of unique column values with no duplicates. If you are unsure, define an index and test it using RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS.

Sorting

You should also note the presence of any notation that indicates the optimizer is performing a sort. Sorting rows before delivery usually slows down query execution time. By defining the appropriate index, you can prevent row sorting, thus speeding up query execution time.

Unless a query includes the ORDER BY clause, the optimizer does not guarantee a specific row order. The optimizer simply finds all the rows that satisfy the query. If you need to specify a particular order for any column, always include an ORDER BY clause in your query.

Avoid descending sorts if no descending indexes exist, because the sort will require an extra step. The order depends on which columns, if any, are indexed. When you define an index for a column, Oracle Rdb arranges the nodes in the index in ascending order of value. Therefore, the default sort order for displayed indexed columns is ASCENDING.

Checking Query Cost

You can check the cost of a query to determine how many I/O operations it requires by using the RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS statistics display (O flag). See Section C.4 for details. This information can be useful when you are developing a very complex query that joins many tables. You can experiment with different forms of the query by defining indexes for particular columns named in the select expression, or by separating the query into smaller and simpler queries. Then, you can run the query using the S and O flags and compare the access strategies the query optimizer chooses for each form of the query with the cost. Use the form of the query that displays the lowest relative cost. Usually the best solution is to simplify the query as much as possible.

In general, you should be concerned with query execution cost only if a particular query poses a problem. Because the database can change considerably from one execution of a query to the next, the optimizer may choose a different access strategy for each execution. Special cases involving complex select expressions, however, may benefit from this kind of analysis before you include them in your host language programs.

5.8.8 Using Query Cost Estimates

You can access optimizer-generated information on the cost of a query and use these cost estimates within an application or interactively. The returned estimates indicate how many I/O operations the query will require and how many rows will be delivered. These cost values are *estimates*, and are identical to the values returned by defining RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS as O.

When you enter the interactive SQL statement SET QUERY CONFIRM and then execute a query, SQL displays the cost of that query and then asks if you wish to cancel the query. This is shown in Example 5–4.

Example 5–4 Using the SET QUERY CONFIRM Statement

```
SQL> SET QUERY CONFIRM
SQL> SELECT   * FROM  EMPLOYEES
cont>   WHERE EMPLOYEE_ID > '01100';
Estimate of query cost:  567 I/Os, rows to deliver:  14
Do you wish to cancel this query (No)?
```

If the cost appears excessive, enter YES and press Return to cancel the query. To continue query execution and display query results, press Return.

The optimizer also writes query cost information to the SQL Communications Area (SQLCA) when you open a cursor. Example 5–5 shows the select expression used in Example 5–4 used within an interactive SQL DECLARE CURSOR statement.

Example 5–5 Accessing Cost Estimates Through the SQLCA

```
SQL> DECLARE CEMP TABLE CURSOR
cont> FOR
cont>     SELECT   *
cont>       FROM  EMPLOYEES
cont>       WHERE EMPLOYEE_ID > '01100';
```

(continued on next page)

Example 5–5 (Cont.) Accessing Cost Estimates Through the SQLCA

```
SQL> OPEN CEMP;
SQL> SHOW SQLCA
SQLCA:
        SQLCAID:        SQLCA           SQLCABC:        128
        SQLCODE:        0
        SQLERRD:        [0]: 0
                        [1]: 0
                        [2]: 35
                        [3]: 29
                        [4]: 0
                        [5]: 0
        SQLWARN0:               SQLWARN1:               SQLWARN2:
        SQLWARN3:               SQLWARN4:               SQLWARN5:
        SQLWARN6:               SQLWARN7:
        SQLSTATE:       00000
SQL>
```

The cost estimates appear in the SQLERRD array.

- SQLERRD element [2] contains the estimated number of rows.
- SQLERRD element [3] contains the estimated number of I/O operations.

If the optimizer cannot provide estimates, the value –1 is returned to indicate "unknown."

To access the estimates using SQL$PRE or SQL$MOD, you must use the QUERY_ESTIMATES qualifier on the command line.

For details on using SQLCA information in applications, refer to the *Oracle Rdb7 Guide to SQL Programming* and the *Oracle Rdb7 SQL Reference Manual*.

Note

To use the new SQLCA array elements in precompiled or module language programs, you must recompile.

If your application tries to access the SQLCA information in SQLERRD elements [2] and [3] on a remote system that is running a version of Oracle Rdb lower than 4.1, the query may not execute.

5.8.9 Constraint BLR May Not Reflect Actual Execution Strategy

When Oracle Rdb compiles a primary or foreign key constraint query, the constraint handling code in Oracle Rdb in most cases reformats the query structure to enable the optimizer to generate the most efficient strategy for constraint evaluation. The reformatting of a constraint query is internal to Oracle Rdb and therefore is not visible to the user or to other software products, such as Oracle Expert for Rdb.

As a result of internal reformatting of a constraint query, the actual execution strategy used to evaluate the constraint may be much different from what it would have been if no internal reformatting was done. The constraint binary language representation (BLR) simply shows the standard format of a primary or foreign key constraint query. It does not reflect any internal reformatting performed by Oracle Rdb. Therefore, you cannot draw conclusions from a constraint BLR about the execution strategy used by Oracle Rdb to evaluate the constraint.

5.8.10 Miscellaneous Hints

The following list provides other hints to help maximize optimizer performance:

- Dynamic optimization depends on the table cardinality estimate (RDB$CARDINALITY column in RDB$RELATIONS system table). This estimate must not deviate much from its true value. Refer to Section 5.3.1 for information on correcting cardinality values.
- Use ANY or EXISTS rather than COUNT if you are attempting to check an occurrence of a specific data value. ANY and EXIST stop as soon as a true condition exists, while COUNT must read all the rows in the Select expression (or at least read all the index values).
- Avoid using NOT, CONTAINING, or MATCHING because the selectivity factor is low. The index may or may not be used, depending on the remainder of the query.
- Do not over index in a heavy update environment. The biggest reason not to over index is the length of time involved in storing a new row, deleting a row, and modifying the key value of an existing row. An additional hindrance caused by updating indexes, is locking at the index level.

  However, in a read-only environment, proliferation of indexes does not hurt, and actually improves, optimization. The only limiting factors are the disk space utilization, the initial table load, and a possible slowdown of the static optimization process.
- Avoid using VARCHAR data type columns in index segments. The optimizer is unable to do index only retrieval for these index segments.

- Do not overuse views or subqueries. This overuse keeps the optimizer from doing its job. Both views and subqueries have their place. If you have questions, try different strategies and use the debug flags to gather information as to which solution fits your data and application.

5.9 Ensuring Query Stability, Controllability, and Performance with Query Outlines

With versions of Oracle Rdb prior to V6.0, there was no direct way to control the strategy the optimizer selected for a query. This meant that a lack of stability and controllability was inherent in the query optimization process.

Stability refers to the maintenance of predictable performance for a query across releases of Oracle Rdb. Changes are made to the optimizer with each Oracle Rdb release. Sometimes these changes to the optimizer can cause a small number of queries that performed acceptably with one version of Oracle Rdb to perform worse with a newer version of Oracle Rdb.

Controllability refers to the ability to manually specify the join order, join methods, and index usage for a query. With versions of Oracle Rdb prior to V6.0, a query that performed unacceptably could be rewritten in the hope that the changes to the query would influence the optimizer to use a different join order, join method, or index when optimizing the query. However, because it was not possible to *directly* specify the join order, join method, or index the optimizer should use, the time-consuming process of rewriting a query did not always bring about the desired improvement in performance.

Beginning with Oracle Rdb V6.0, you can directly control the strategy the optimizer selects for a query by defining an outline for the query. A **query outline** is an overall plan for how a query can be implemented. Outlines originate from the query optimizer and can be extracted, edited, stored, used, or ignored. Outlines can be defined that contain directives that control the join order, join methods, or index usage (or all of these) the optimizer selects when processing a query.

Outlines can be used to solve the stability and controllability problems that were inherent in the query optimization process prior to Oracle Rdb V6.0. You can obtain cross-release stability for a query by storing and reusing outlines from a previous Oracle Rdb version. You can obtain controllability by editing outlines to directly control the solution generated by the optimizer. In some cases, you can obtain improved run-time performance by manually improving the join order, join methods, or index selection decisions made by the optimizer.

Outlines should be used sparingly. The retrieval strategies chosen by the optimizer result in good performance for most queries; outlines should be used only for the small percentage of queries that perform poorly due to poor strategy decisions made by the optimizer, or when cross-release stability is critical. End users and application programmers should not define query outlines. The database administrator (DBA) or any other person experienced in examining database performance issues should examine a query that is considered a candidate for a query outline. In some cases (especially with new queries), the performance of a query may be poor not because the optimizer is choosing the wrong strategy, but because the query is written poorly. In such a case, the solution is to rewrite the query so that it retrieves data more economically, not to define an outline.

When an outline is needed, it should be defined by a DBA or another user experienced in database performance issues. The optimizer selects retrieval strategies based on the information available to it; in some cases a person experienced in database performance has more knowledge than the optimizer about data interrelationships, data distribution, or the partitioning of data, and therefore can select a better strategy than the optimizer's choice.

You can specify partial or complete outlines. Partial outlines enable the DBA to specify a few critical decisions and have the optimizer fill in the remaining details, thereby minimizing the effort required to generate the globally optimal query plan. Complete outlines provide complete control over join order, join methods, and index usage for a query. Both partial and complete outlines provide a means of ensuring that the optimizer generates the optimal plan for every query; users do not need to resort to ad hoc modifications of queries. Section 5.9.4.2 describes complete outlines, and Section 5.9.4.3 describes partial outlines.

It is also possible to create multiple outlines for the same query. This is useful for addressing criteria that are not considered by the optimizer. For example, consider a query that is executed during the day and at night. During the day, when contention for resources is expected to be high, a different retrieval solution may be appropriate for the query than at night, when there is little or no resource contention. Section 5.9.4.1 describes how to create multiple outlines for a query.

Outlines are most useful for large, critical applications and for very large databases (in terabyte-sized databases, the effect of suboptimal query execution is magnified).

Oracle Rdb provides the **Query Performance Tuner (QPT)**, a Windows interface for creating and editing outlines. QPT also allows you to tune individual queries for maximum performance. QPT generates a graphical model of the optimization strategy for an SQL query, and enables you to modify any aspect of the solution (join order, access paths, join methods, execution strategy). The strategy may then be saved in the database and will be applied on subsequent compilations and executions of the query. See the Windows help for information about using QPT.

Section 5.9.1 through Section 5.9.9 describe how to create, modify, delete, and use outlines.

5.9.1 Using Optimizer Output to Define an Outline to Be Stored

You define a new outline by using query outlines generated by the Oracle Rdb optimizer.

OpenVMS VAX OpenVMS Alpha

Example 5–6 shows how to use the RDMS$DEBUG_FLAGS and RDMS$DEBUG_FLAGS_OUTPUT logical names on OpenVMS to capture outlines generated by the Oracle Rdb optimizer in an output file. The query in the example displays information on the educational degrees earned by all employees over 65 years old.

Example 5–6 Capturing Outlines Generated by the Optimizer

```
$! Define RDMS$DEBUG_FLAGS to "Ss" so that outlines generated
$! by the optimizer are displayed.
$ DEFINE RDMS$DEBUG_FLAGS "Ss"
$!
$! Define RDMS$DEBUG_FLAGS_OUTPUT to be a file that will contain
$! the outlines generated during the session.
$ DEFINE RDMS$DEBUG_FLAGS_OUTPUT degrees_for_emps_over_65.sql
$ SQL
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SELECT E.LAST_NAME, E.FIRST_NAME, E.EMPLOYEE_ID, D.DEGREE, D.YEAR_GIVEN
cont>   FROM EMPLOYEES E, DEGREES D
cont>   WHERE E.BIRTHDAY < '31-Dec-1928'
cont>         AND E.EMPLOYEE_ID = D.EMPLOYEE_ID
cont>   ORDER BY E.LAST_NAME;
 E.LAST_NAME      E.FIRST_NAME   E.EMPLOYEE_ID   D.DEGREE   D.YEAR_GIVEN
 Babbin           Joseph         00207           MA                 1980
 Babbin           Joseph         00207           MA                 1980
 Bartlett         Dean           00173           BA                 1978
 Clairmont        Rick           00231           BA                 1967
 Herbener         James          00471           BA                 1981
 Herbener         James          00471           MA                 1982
```

(continued on next page)

Example 5–6 (Cont.) Capturing Outlines Generated by the Optimizer

```
 Johnson          Bill           00240           BA                 1970
 Kinmonth         Louis          00177           BA                 1982
 Nash             Walter         00183           BA                 1977
 Nash             Walter         00183           PhD                1978
 O’Sullivan       Rick           00190           BA                 1982
 O’Sullivan       Rick           00190           MA                 1983
 Reitchel         Charles        00193           BA                 1982
 Ziemke           Al             00200           MA                 1971
 Ziemke           Al             00200           MA                 1971
15 rows selected
SQL> ROLLBACK;
SQL> DISCONNECT DEFAULT;
SQL> EXIT;
$!
$! Display the output file generated by the optimizer:
$ TYPE degrees_for_emps_over_65.sql
Leaf#01 FFirst RDB$RELATIONS Card=19
  BgrNdx1 RDB$REL_REL_NAME_NDX [1:1] Fan=8
-- Rdb Generated Outline :  8-JUN-1993 14:19  ❶
create outline QO_982C2D52C1D95DA2_00000000
id ’982C2D52C1D95DA2F46F0A7090B28309’
mode 0
as (
  query (
    subquery (
      RDB$RELATIONS 0         access path index       RDB$REL_REL_NAME_NDX
      )
    )
  )
compliance optional     ;
Sort
Cross block of 2 entries
  Cross block entry 1
    Leaf#01 BgrOnly RDB$RELATION_FIELDS Card=141
      BgrNdx1 RDB$RFR_REL_NAME_FLD_ID_NDX [1:1] Fan=8
  Cross block entry 2
    Get     Retrieval by index of relation RDB$FIELDS
      Index name  RDB$FIELDS_NAME_NDX [1:1]  Direct lookup
-- Rdb Generated Outline :  8-JUN-1993 14:19  ❷
create outline QO_E8DB158FDFBD61CD_00000000
id ’E8DB158FDFBD61CDA331711179A010E6’
mode 0
as (
  query (
    subquery (
      RDB$RELATION_FIELDS 0   access path index
```

(continued on next page)

Example 5–6 (Cont.) Capturing Outlines Generated by the Optimizer

```
      RDB$RFR_REL_NAME_FLD_ID_NDX
        join by cross to
      RDB$FIELDS 1    access path index       RDB$FIELDS_NAME_NDX
      )
    )
  )
compliance optional     ;
Cross block of 2 entries
  Cross block entry 1
    Conjunct        Get     Retrieval by index of relation EMPLOYEES
      Index name  EMP_LAST_NAME [0:0]
  Cross block entry 2
    Leaf#01 FFirst DEGREES Card=165
      BgrNdx1 DEG_EMP_ID [1:1] Fan=17
-- Rdb Generated Outline :  8-JUN-1993 14:19  ❸
create outline QO_284D6F269B44A56F_00000000
id '284D6F269B44A56F6C2BC8998832FD1D'
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMP_LAST_NAME
        join by cross to
      DEGREES 1       access path index       DEG_EMP_ID
      )
    )
  )
compliance optional     ;
♦
```

You can also use the RDB_DEBUG_FLAGS and RDB_DEBUG_FLAGS_OUTPUT configuration parameters on Digital UNIX to capture outlines. Note that you must specify "Ss" (an uppercase S followed by a lowercase s) with the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter to display outlines generated by the optimizer.

In Example 5–6, the output file contains three generated outlines. Each outline is preceded by the SQL comment string "– Rdb Generated Outline" and the creation time, and terminates with a semicolon (;).

The following callouts are keyed to Example 5–6:

❶ The first outline generated by the optimizer shows the strategies the optimizer chose when accessing system tables. This is not the outline the optimizer generated for the SELECT statement.

❷ The second outline generated by the optimizer also shows the strategies the optimizer chose when accessing system tables. This is not the outline the optimizer generated for the SELECT statement.

❸ The third outline is the outline the optimizer generated for the SELECT statement. This outline, which shows the EMPLOYEES and DEGREES tables and EMP_LAST_NAME and DEG_EMP_ID indexes that the optimizer uses when processing the query, is the generated outline to store in the database.

In Oracle Rdb, views are implemented as stored queries. When a query uses a view, the outline generated for the query expands the view reference into a subquery that shows the tables that comprise the view.

OpenVMS VAX OpenVMS Alpha

Example 5–7 shows how views are represented in outlines that are generated by the optimizer. The query in Example 5–7 uses the CURRENT_JOB view and the EMPLOYEES table to display information on employees over 65 years old that have been in the same job since January, 1982.

Example 5–7 Representation of Views in Outlines Generated by the Optimizer

```
$! Define RDMS$DEBUG_FLAGS to "Ss" so that outlines generated by
$! the optimizer are displayed.
$ DEFINE RDMS$DEBUG_FLAGS "Ss"
$!
$! Define RDMS$DEBUG_FLAGS_OUTPUT to be a file that will contain
$! the outlines generated during the session.
$ DEFINE RDMS$DEBUG_FLAGS_OUTPUT over_65_10_years_in_job.sql
$ SQL
SQL> ATTACH 'FILENAME mf_personnel';
SQL> --
SQL> SELECT CJ.EMPLOYEE_ID, CJ.LAST_NAME, CJ.FIRST_NAME,
cont>   E.BIRTHDAY, CJ.JOB_START
cont>   FROM CURRENT_JOB CJ, EMPLOYEES E
cont>   WHERE CJ.EMPLOYEE_ID = E.EMPLOYEE_ID AND CJ.JOB_START > '31-Dec-1981'
cont>         AND E.BIRTHDAY < '31-Dec-1928';
 CJ.EMPLOYEE_ID   CJ.LAST_NAME     CJ.FIRST_NAME   E.BIRTHDAY    CJ.JOB_START
 00190            O'Sullivan       Rick            12-Jan-1923   25-Feb-1982
1 row selected
SQL> --
SQL> COMMIT;
SQL> DISCONNECT DEFAULT;
SQL> EXIT
$ DEASSIGN RDMS$DEBUG_FLAGS
$ DEFINE RDMS$DEBUG_FLAGS_OUTPUT
```

(continued on next page)

Example 5–7 (Cont.) Representation of Views in Outlines Generated by the Optimizer

```
$!
$! Display the section of the over_65_10_years_in_job.sql output
$! file that was generated for the query:
$ TYPE over_65_10_years_in_job.sql
   .
   .
   .
Cross block of 2 entries
  Cross block entry 1
    Conjunct        Get     Retrieval by index of relation EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:0]
  Cross block entry 2
    Cross block of 2 entries
      Cross block entry 1
        Get     Retrieval by index of relation EMPLOYEES
          Index name  EMPLOYEES_HASH [1:1]       Direct lookup
      Cross block entry 2
        Conjunct        Conjunct        Get
        Retrieval by index of relation JOB_HISTORY
          Index name  JOB_HISTORY_HASH [1:1]
-- Rdb Generated Outline : 22-JUN-1993 16:22
create outline QO_B0B5EFA1486E0447_00000000
id 'B0B5EFA1486E0447C3B41C6E842558B6'
mode 0
as (
  query (
    subquery (
      EMPLOYEES 2     access path index       EMP_EMPLOYEE_ID
        join by cross to
      subquery (
        EMPLOYEES 1     access path index       EMPLOYEES_HASH    ❶
          join by cross to
        JOB_HISTORY 0   access path index       JOB_HISTORY_HASH  ❷
        )
      )
    )
  )
compliance optional     ;
$
♦
```

The following callouts are keyed to Example 5–7:

❶ The EMPLOYEES table is used in the definition of the CURRENT_JOBS view.

❷ The JOB_HISTORY table is used in the definition of the CURRENT_JOBS view.

Because the outlines generated by the optimizer are already in extended SQL CREATE OUTLINE statement format, you can use a text editor to remove all the text from the output file except the outline you are interested in, and then use the edited output file as an SQL command procedure to store the outline (assuming your output file has a file type of .sql). In Example 5–8, all the text is removed from the degrees_for_emps_over_65.sql file except the third generated outline. When you are editing the output file to remove extraneous text, you can also supply a more meaningful outline name than the outline name generated by the optimizer. For example, you could change the outline name for the third generated outline in Example 5–6 to degrees_for_emps_over_65, resulting in the edited output file shown in Example 5–8.

Example 5–8 Changing the Outline Name Generated by the Optimizer

```
-- Rdb Generated Outline :  8-JUN-1993 14:19
create outline DEGREES_FOR_EMPS_OVER_65  ❶
id '284D6F269B44A56F6C2BC8998832FD1D'    ❷
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0  ❸   access path index       EMP_LAST_NAME
        join by cross to
      DEGREES 1    ❸   access path index       DEG_EMP_ID
      )
    )
  )
compliance optional
comment is 'Created by Fred Smith' ;  ❹
```

The following callouts are keyed to Example 5–8:

❶ The generated outline name (QO_284D6F269B44A56F_00000000 from Example 5–6) has been changed to the more meaningful outline name degrees_for_emps_over_65.

❷ Be sure *not* to modify the outline id when you edit the file. If you do, the association between the outline and the original query is destroyed, and the outline will be unusable after it is stored.

❸ Do not modify the table instance numbers that the optimizer generates for tables used in the query. If a table appears more than once in an outline, a unique table instance number is generated for each reference to the table in the outline. The table instance number is 0 for the EMPLOYEES table. The table instance number is 1 for the DEGREES table.

❹ You can add the COMMENT IS clause to provide a comment for the outline.

Section 5.9.4 describes the modifications that you can make to a generated outline.

You store the outline for the query by attaching to the database and executing the degrees_for_emps_over_65.sql command file from SQL. After it is stored, the outline can be displayed with the SHOW OUTLINES statement, as shown in Example 5–9.

Example 5–9 Storing an Outline Generated by the Optimizer

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> @degrees_for_emps_over_65.sql
SQL> --
SQL> -- Display the outline using the SHOW OUTLINES statement:
SQL> SHOW OUTLINES degrees_for_emps_over_65
     DEGREES_FOR_EMPS_OVER_65
 Comment:       Created by Fred Smith
 Source:

create outline DEGREES_FOR_EMPS_OVER_65
id '284D6F269B44A56F6C2BC8998832FD1D'
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMP_LAST_NAME
        join by cross to
      DEGREES 1       access path index       DEG_EMP_ID
      )
    )
  )
compliance optional     ;
```

When you define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter to be "Ss" and the RDMS$DEBUG_FLAGS_OUTPUT logical name or the RDB_DEBUG_FLAGS_OUTPUT configuration parameter to be an output file, the outline that the optimizer generates for each database query is written to the output file. The optimizer generates outlines for any front-end product (such as Oracle Rally [1]) that can query an Oracle Rdb database. Outlines that are generated for a layered product accessing the database can be stored in the database in the same way as outlines generated by SQL statements.

[1] On OpenVMS systems only

Outlines that are stored for a view are not applied when that view is embedded in another query.

Outlines generated by the optimizer are in extended SQL CREATE OUTLINE statement format. To store an outline, you must have the SQL CREATE privilege for every table referenced by the outline. The CREATE OUTLINE statement is an online operation (other users can be attached to the database when an outline is created).

5.9.2 Specifying Outline Directives

By using the SQL CREATE OUTLINE statement, you can define outlines that contain directives that control decisions the optimizer makes when processing a query. The following list is an overview of the outline directives that can be specified with a CREATE OUTLINE statement. The complete syntax for these outline directives can be found in the CREATE OUTLINE statement description in the *Oracle Rdb7 SQL Reference Manual*.

- Join order

  The join order is the order in which table instances, view instances, or subqueries (join items) are joined together during optimization. With an outline, you can specify:

  - The order that the optimizer will impose for the groupings of join items accessed during the query.
  - That the optimizer will not impose an order on the groupings of join items accessed during the query.
  - That the optimizer will not impose an order on specific join items within the same subquery level. When the optimizer is allowed to group certain join items at the same level in any order, those join items are said to be *floating*.

  Although the CREATE OUTLINE syntax lets you specify join items in different orders, the optimizer may not be able to use the join order specified in an outline. For example, contexts in a subquery that need values from an outer context cannot be placed before the outer context in the join order. If this occurs, the optimizer will not be able to use the join strategy specified in the outline.

- Join method

  The join method is the algorithm used by the optimizer to associate records in a join. In an outline, you can specify that the optimizer use the cross join strategy, match join strategy, union strategy, or any method to join two data sources.

Although there are several syntactically valid join methods that can be specified with the CREATE OUTLINE statement, the optimizer may not be able to use the join method specified in the outline. For example:

  - The match join strategy requires that an equivalent join column exist between the inner and outer context of the join order. If the query for which the outline is created does not have an equivalent join column, then the optimizer cannot use the match join strategy specified in the outline.
  - The union strategy is valid only for queries that use the UNION operator, and all queries that specify the UNION operator must use the union strategy.

- Access path

  The access path is the method used to retrieve table rows. You can specify that the optimizer return rows by database key (dbkey), sequential reads, index (you can specify the index you want the optimizer to use), without using an index, or by any method.

  You can create a syntactically valid outline that specifies an access path that the optimizer cannot use. For example, if you specify access by dbkey when the dbkeys are not available, the optimizer cannot use the outline's specified access path. See Example 5–14 for an example of a syntactically valid outline that specifies an access path that the optimizer cannot use.

- Compliance level

  An outline can be defined to have a compliance level of mandatory or optional.

  When the compliance level for an outline is defined as mandatory, all the outline directives (such as the join order, join methods, and index usage) must be followed as outlined. See Section 5.9.4.4 for more information on mandatory outlines.

  When the compliance level for an outline is defined as optional, all the outline directives are optional. If the optimizer cannot follow all the directives specified in an optional compliance outline, it can select other strategies to process the query. See Section 5.9.4.5 for more information on optional outlines.

- Execution options

  These are dynamic optimization options that the optimizer should take into account at run time. The valid execution options are FAST FIRST, TOTAL TIME, ANY, and NONE.

The FAST FIRST execution option specifies that the optimizer is permitted to use fast first dynamic optimization when appropriate. In some cases, fast first retrieval cannot be used. For example, total time retrieval must be used if an aggregate expression (AVG, COUNT, MAX, MIN, and SUM) is used without a GROUP BY clause in the query.

The TOTAL TIME execution option specifies that the optimizer is permitted to use total time dynamic optimization when appropriate. In some cases, total time retrieval cannot be used. For example, fast first retrieval must be used if the EXISTS predicate is used in the query.

The ANY option specifies that the optimizer is free to choose any optimization method.

The NONE option specifies that no optional run-time optimization be used. Specifying the NONE option causes the query to be very stable across versions of Oracle Rdb. However, this stability is likely to come at the expense of performance.

The default execution option is ANY, which means that dynamic optimization is enabled and any dynamic optimization strategy can be used.

See the CREATE OUTLINE statement description in the *Oracle Rdb7 SQL Reference Manual* for more information on the syntax for specifying outline directives.

5.9.3 Defining and Storing an Outline for a Stored Procedure

When a stored procedure has been stored in the database, you can use the CREATE OUTLINE statement to define and store an outline for that stored procedure. The *Oracle Rdb7 Guide to SQL Programming* describes how stored procedures are defined and stored in a database.

Example 5–10 shows how to use the SQL CREATE OUTLINE statement with the ON PROCEDURE NAME clause to define an outline for a stored procedure.

Example 5–10 Defining an Outline for a Stored Procedure

```
SQL> -- Store the procedure in the database:
SQL> CREATE MODULE my LANGUAGE sql AUTHORIZATION rick
cont> --
```

(continued on next page)

Example 5–10 (Cont.) Defining an Outline for a Stored Procedure

```
cont> PROCEDURE p1 (:x CHAR(14),  :y SMALLINT);
cont>    BEGIN
cont>    SELECT last_name INTO :x FROM employees LIMIT TO 1 ROW;
cont>    END;
cont> END MODULE;
SQL>
SQL> -- Create an outline for the new stored procedure:
SQL> CREATE OUTLINE LAST_NAME_OUTLINE ON PROCEDURE NAME P1
cont> MODE -1 COMPLIANCE OPTIONAL;
SQL> SHOW OUTLINES LAST_NAME_OUTLINE
     LAST_NAME_OUTLINE
 Source:

-- Rdb Generated Outline : 21-DEC-1993 09:15
create outline LAST_NAME_OUTLINE
id 'F7542B8932D2B8232131DCB52EAE205F'
mode -1
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMP_LAST_NAME
      )
    )
  )
compliance optional     ;
SQL>
```

The procedure-id must be a valid procedure id for an existing stored procedure already stored in the database. See the *Oracle Rdb7 SQL Reference Manual* for the complete syntax for the CREATE OUTLINE statement.

5.9.4 Modifying an Existing Outline

You can modify an existing outline by using the RMU Extract command and the SQL CREATE OUTLINE statement.

OpenVMS VAX ≡ OpenVMS Alpha ≡

Use the RMU Extract command to extract the outlines for a database into an output file, as shown in Example 5–11.

Example 5–11 Extracting Existing Outlines into a File

```
$ RMU/EXTRACT/ITEM=(OUTLINES)/LANGUAGE=SQL -
_$ /OUTPUT=degrees_for_emps_over_65.sql mf_personnel
$!
$! Show the RMU Extract output in degrees_for_emps_over_65.sql:
$ TYPE degrees_for_emps_over_65.sql
-- RMU/EXTRACT for Oracle Rdb V6.0-0                     8-JUN-1993 17:14:36.21
--
--                           Database Definition File
--
-- Source Database Name: SQL_DISK1:[RICK.V60]MF_PERSONNEL.RDB;1
--------------------------------------------------------------------------------
set verify
set language ENGLISH
set default date format 'SQL92';
set quoting rules 'SQL92';
set date format DATE 001, TIME 001
attach 'pathname MF_PERSONNEL';

-- RMU/EXTRACT for Oracle Rdb V6.0-0                     8-JUN-1993 17:14:36.21
--
--                          Query Outline Definitions
--
--------------------------------------------------------------------------------
create outline DEGREES_FOR_EMPS_OVER_65
id '284D6F269B44A56F6C2BC8998832FD1D'
mode 0
as (
    query (
        subquery (
            EMPLOYEES 0 access path index EMP_LAST_NAME
                join by cross to
            DEGREES 1 access path index DEG_EMP_ID
            )
        )
    )
compliance optional
execution options (any);

$
```

♦

As shown in Example 5–11, the RMU Extract command extracts outlines in CREATE OUTLINE statement format. Edit the output file (degrees_for_emps_over_65.sql in our example) to remove any text not associated with the outline to be modified and to make the desired changes to the outline.

When the edited output file contains a CREATE OUTLINE statement with the desired modifications to the existing outline, attach to the database and delete the existing outline (using the SQL DROP OUTLINE statement shown in Section 5.9.9). Because each outline stored in a database must have a unique name and outline mode, the modified version of the outline that has the same outline name and outline mode as the existing version cannot be stored until the existing outline is deleted.

After deleting the existing outline, you can store the modified outline by attaching to the database and executing the modified output file as an SQL command procedure.

See the *Oracle RMU Reference Manual* for more information on the RMU Extract command.

5.9.4.1 Creating Multiple Outlines for a Single Query

Because the optimizer cannot consider all criteria when it generates a solution to a query, sometimes you may want to define more than one outline for a query. For example, the optimizer cannot determine whether contention for resources is high or low when it processes a query. It is not unusual for some queries to execute much faster at night (when there is less contention for resources) than during the day. For this situation, the DBA can create one outline for daytime processing and another outline for nighttime processing.

If contention is high for the EMPLOYEES and DEGREES tables during the day, and lower at night, you can define a second outline as an alternative to the one defined in Example 5–9 for use at night. After extracting the degrees_for_emps_over_65 outline into the output file degrees_for_emps_over_65.sql as shown in Example 5–11, you can modify the extracted outline to create a definition for another outline that will be used for the same query.

If the query is part of a batch job that runs at night, you could change the original degrees_for_emps_over_65 outline to require the total time method of retrieval, which might improve performance of the query at night. Fast first retrieval (the method selected by the optimizer for the original degrees_for_emps_over_65 outline) optimizes retrieval time for the first few records that satisfy a query. Fast first optimization is a good choice if a query will be executed interactively and it is likely to be terminated before all the records satisfying the query are returned (by a user pressing Ctrl/C, for example).

When the query is run at night as part of a batch job, the query is not likely to be stopped until all the records are retrieved. Because the total time retrieval method optimizes total retrieval time, it could be the most appropriate retrieval method for this query when it is run at night. This change is made by changing “execution options (any)” in the extracted outline to “execution options (total time)”. See Section 5.9.2 for more information on specifying retrieval methods.

When you are defining an outline for a query for which one or more other outlines are stored in the database, the outline being defined must have a unique outline name and outline mode. The existing outline in this example has an outline name of degrees_for_emps_over_65 and the default outline mode of 0, as shown in Example 5–11. An outline mode is a signed integer; valid values for the outline mode are –2,147,483,648 to 2,147,483,647. The new outline for the query is given the outline name degrees_for_emps_over_65_night and an outline mode of –1, as shown in Example 5–12. Positive mode values are reserved for future use by Oracle Corporation, so it is recommended that you specify a value betweeen 0 and –2,147,483,648 for the mode value. The outline mode value you specify for an outline should be a value that is shared by other outlines that will be used at the same time. For example, you could decide that any outline that will run only at night should be given an outline mode value of –1. In Example 5–12, the degrees_for_emps_over_65_night outline is given an outline mode value of –1 for this reason.

After all the changes are made, the modified output file, degrees_for_emps_over_65.sql, looks like this:

```
create outline DEGREES_FOR_EMPS_OVER_65_NIGHT
id '284D6F269B44A56F6C2BC8998832FD1D'
mode -1
as (
    query (
        subquery (
            EMPLOYEES 0 access path index EMP_LAST_NAME
                join by cross to
            DEGREES 1 access path index DEG_EMP_ID
            )
        )
    )
compliance optional
execution options (TOTAL TIME);
```

The degrees_for_emps_over_65.sql file can be used to store the outline, as shown in Example 5–12.

Example 5–12 Creating Multiple Outlines for a Query

```
SQL> --
SQL> -- The output file is used as an SQL command procedure to define
SQL> -- and store the outline:
SQL> @DEGREES_FOR_EMPS_OVER_65.SQL
SQL> --
```

(continued on next page)

Example 5–12 (Cont.) Creating Multiple Outlines for a Query

```
SQL> -- Display the outline:
SQL> SHOW OUTLINES DEGREES_FOR_EMPS_OVER_65_NIGHT
     DEGREES_FOR_EMPS_OVER_65_NIGHT
 Source:

create outline DEGREES_FOR_EMPS_OVER_65_NIGHT
id ’284D6F269B44A56F6C2BC8998832FD1D’
mode -1
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMP_LAST_NAME
        join by cross to
      DEGREES 1       access path index       DEG_EMP_ID
      )
    )
  )
compliance optional
execution options (      total time )   ;
```

Outlines support multi-octet characters in:

- Outline names
- Index and table names specified in the outline
- Descriptions and comments specified in the outline

The section on character sets in the *Oracle Rdb7 SQL Reference Manual* provides more information about multi-octet characters.

5.9.4.2 Complete Outlines

In a complete outline, the outline directives specify everything about the strategy the optimizer should use for every table and join combination in the query. That is, a complete outline specifies each of the directives described in Section 5.9.2.

The outlines generated by the optimizer are always complete outlines.

Complete outlines are especially useful to ensure that the performance of a query remains stable from one release of Oracle Rdb to another, despite changes made to the optimizer. With a complete outline, the optimizer is constrained to always select the join orders, access paths, join methods, and execution options specified by the outline.

The optimizer can follow all directives in a complete outline only when all tables and indexes specified in the outline exist in the database. If an index or table specified in a complete outline is deleted, Oracle Rdb marks the outline as invalid and the optimizer is not able to follow all the directives in the outline when the query is processed. See Section 5.9.8 for information on invalidated outlines.

5.9.4.3 Partial Outlines

In a partial outline, the outline directives do not specify everything about the strategy the optimizer should use for every table and join combination in the query. That is, a partial outline does not specify each of the directives described in Section 5.9.2.

A partial outline affords a degree of freedom to the optimizer so it can choose alternatives. This means that a partial outline is less likely to become invalid than a complete outline. An outline is partial if:

- The ANY option is specified for the join method, access path, or execution options
- The FLOATING or UNORDERED clause is used anywhere in the outline definition
- Any subquery, join, or table that takes part in the query is omitted from the outline

In some cases, a partial outline can be better than a complete outline. In a partial outline, it is most important to correctly specify collections of join items that the optimizer should keep together, and whether those join items should be ordered or unordered. You can avoid overconstraining the optimizer by specifying a join method of ANY, an access path of ANY, and omitting tables that are not critical to the solution. This allows the optimizer to adjust for changes in data distribution and run-time variable bindings.

When defining an outline for a query, you can refine the outline in steps. The first step is to constrain the start of the join order, letting the optimizer figure out the details. You can then make further refinements to the outline if this first step does not result in acceptable run-time performance for the query after the outline has been stored.

With a partial outline, the noncritical elements of the outline are not specified. If you modify a complete outline by removing references to noncritical tables and indexes, the resulting partial outline has less dependence on specific tables and indexes than the previous complete outline. This means that if a noncritical index or table is deleted from the database, the new partial outline will not be invalidated by the deletion. However, as with complete outlines, if a table or index that *is* specified in a partial outline is deleted, Oracle Rdb marks the outline as invalid and cannot use it when the query is processed. See Section 5.9.8 for information on invalidated outlines.

See the CREATE OUTLINE statement description in the *Oracle Rdb7 SQL Reference Manual* for the complete syntax for specifying outline directives.

5.9.4.4 Mandatory Outlines

For outlines defined with a compliance level of mandatory, the optimizer is required to follow all of the outline directives such as the join order, join methods, and index usage. If the optimizer cannot follow all of the outline directives in a mandatory outline, the query fails.

OpenVMS VAX OpenVMS Alpha

Example 5–13 shows the error message that is displayed when a query fails.

Example 5–13 Error Message Displayed When the Optimizer Cannot Follow a Directive in a Mandatory Outline

```
$! Define the appropriate logical names so that outlines generated
$! by the optimizer can be logged to an output file.
$ DEFINE RDMS$DEBUG_FLAGS "Ss"
$ DEFINE RDMS$DEBUG_FLAGS_OUTPUT employees_outline.sql
$ SQL$
SQL> ATTACH 'FILENAME mf_personnel';
SQL> --
SQL> -- Execute a query for which an outline will be defined.
SQL> SELECT EMPLOYEE_ID FROM EMPLOYEES WHERE EMPLOYEE_ID BETWEEN
cont> '00150' AND '00175';
 EMPLOYEE_ID
 00164
 00165
 00166
 00167
 00168
 00169
 00170
 00171
 00172
 00173
 00174
 00175
12 rows selected
SQL> COMMIT;
SQL> DISCONNECT DEFAULT;
```

(continued on next page)

Example 5–13 (Cont.) Error Message Displayed When the Optimizer Cannot Follow a Directive in a Mandatory Outline

```
SQL> EXIT;
$ DEASSIGN RDMS$DEBUG_FLAGS_OUTPUT
$!
$! Display the portion of the generated optimizer output for
$! the query for which the outline will be defined.  The output
$! shows that the optimizer used the sorted index EMP_EMPLOYEE_ID
$! to retrieve the range of EMPLOYEE_ID column values specified by
$! the SELECT query.
$ TYPE employees_outline.sql
   .
   .
   .
--  Rdb Generated Outline : 11-JUN-1993 12:15
create outline QO_FAF0A23CF87C0FEE_00000000
id 'FAF0A23CF87C0FEE840D9BCA37A236B5'
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
      )
    )
  )
compliance optional     ;
$!
$! Edit the generated outline, changing the generated outline name
$! for the query to FIND_EMPLOYEE_IDS, the index used to EMPLOYEES_HASH,
$! and the compliance level to mandatory.
$ EDIT employees_outline.sql
   .
   .
   .
$!
$! Display the modified output file.
$ TYPE employees_outline.sql
--  Rdb Generated Outline : 11-JUN-1993 12:15
create outline FIND_EMPLOYEE_IDS
id 'FAF0A23CF87C0FEE840D9BCA37A236B5'
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance mandatory     ;
$ DEASSIGN RDMS$DEBUG_FLAGS
$ !
```

(continued on next page)

Example 5–13 (Cont.) Error Message Displayed When the Optimizer Cannot Follow a Directive in a Mandatory Outline

```
$ ! Define and store the outline, using the modified output file as an
$ ! SQL command procedure:
$ SQL$
SQL> ATTACH ’FILENAME mf_personnel’;
SQL> @employees_outline.sql
SQL> SHOW OUTLINES FIND_EMPLOYEE_IDS
     FIND_EMPLOYEE_IDS
 Source:

create outline FIND_EMPLOYEE_IDS
id ’FAF0A23CF87C0FEE840D9BCA37A236B5’
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance mandatory    ;
SQL> COMMIT;
SQL> --
SQL> -- When the query is issued now, it fails because the outline
SQL> -- directives specified as mandatory cannot be followed (the
SQL> -- hashed index EMPLOYEES_HASH cannot be used for the range
SQL> -- retrieval of EMPLOYEE_ID column values required by the query).
SQL> SELECT EMPLOYEE_ID FROM EMPLOYEES WHERE EMPLOYEE_ID BETWEEN
cont> ’00150’ AND ’00175’;
%RDMS-F-OUTLINE_FAILED, could not comply with mandatory query outline directives
SQL>
♦
```

If your priority for a query is that a particular optimizer strategy always be used when the query is processed, then specifying a compliance level of mandatory for the outline is probably appropriate. If the optimizer cannot follow all of the outline directives, the query will fail and this will tell you that you need to see why the outline is invalid.

5.9.4.5 Optional Outlines

For outlines defined with a compliance level of optional, the optimizer is not required to follow all of the outline directives. If the optimizer cannot follow all directives specified in an optional compliance outline, it can select other strategies to process the query.

Example 5–14 shows that when the optimizer cannot follow one of the outline directives for an outline defined with a compliance level of optional, the optimizer chooses an alternate strategy to process the query.

OpenVMS VAX OpenVMS Alpha

Example 5–14 also shows that for an outline with a compliance level of optional, Oracle Rdb informs you that the outline directives have not been fully complied with only when you have defined the RDMS$DEBUG_FLAGS logical name to be "Ss".

Example 5–14 Determining Whether or Not an Optional Compliance Level Outline Has Been Fully Complied With

```
$! The RDMS$DEBUG_FLAGS logical name is not set to "Ss".
$ SHOW LOGICAL RDMS$DEBUG_FLAGS
%SHOW-S-NOTRAN, no translation for logical name RDMS$DEBUG_FLAGS
$ SQL
SQL> ATTACH 'FILENAME mf_personnel';
SQL> --
SQL> -- Note that the compliance level for the outline is optional
SQL> -- in this example.
SQL> SHOW OUTLINES FIND_EMPLOYEE_IDS
     FIND_EMPLOYEE_IDS
 Source:

create outline FIND_EMPLOYEE_IDS
id 'FAF0A23CF87C0FEE840D9BCA37A236B5'
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance optional     ;
SQL> --
SQL> -- Issue the query for which the outline was defined.  When the
SQL> -- RDMS$DEBUG_FLAGS logical name is not set to "Ss", Oracle Rdb
SQL> -- gives no indication that the directives in the outline
SQL> -- were not fully complied with.
SQL> SELECT EMPLOYEE_ID FROM EMPLOYEES WHERE EMPLOYEE_ID BETWEEN
cont> '00150' AND '00175';
 EMPLOYEE_ID
 00166
 00167
```

(continued on next page)

Example 5–14 (Cont.) Determining Whether or Not an Optional Compliance Level Outline Has Been Fully Complied With

```
 00173
 00169
 00175
 00172
 00164
 00168
 00165
 00171
 00170
 00174
12 rows selected
SQL> COMMIT;
SQL> DISCONNECT DEFAULT;
SQL> EXIT;
$!
$! Define RDMS$DEBUG_FLAGS as "Ss".
$ DEFINE RDMS$DEBUG_FLAGS "Ss"
$ SQL
SQL> ATTACH ’FILENAME mf_personnel’;
SQL> --
SQL> -- Issue the original query again.  Notice that with the
SQL> -- RDMS$DEBUG_FLAGS logical name set to "Ss", Oracle Rdb informs
SQL> -- you that the FIND_EMPLOYEE_IDS outline was used, but full
SQL> -- compliance with the outline was not possible.  The display
SQL> -- also shows that the optimizer used sequential retrieval to
SQL> -- return the requested EMPLOYEE_ID column values (because
SQL> -- the EMPLOYEES_HASH index specified in the outline is not
SQL> -- able to retrieve a range of values).
SQL> SELECT EMPLOYEE_ID FROM EMPLOYEES WHERE EMPLOYEE_ID BETWEEN
cont> ’00150’ AND ’00175’;
   .
   .
   .

~S: Outline FIND_EMPLOYEE_IDS used
~S: Full compliance with the outline was not possible
Conjunct        Get     Retrieval sequentially of relation EMPLOYEES
```

(continued on next page)

Example 5–14 (Cont.) Determining Whether or Not an Optional Compliance Level Outline Has Been Fully Complied With

```
--  Rdb Generated Outline : 11-JUN-1993 13:04
create outline QO_FAF0A23CF87C0FEE_00000000
id 'FAF0A23CF87C0FEE840D9BCA37A236B5'
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path sequential
      )
    )
  )
compliance optional     ;
 EMPLOYEE_ID
 00166
 00167
 00173
 00169
 00175
 00172
 00164
 00168
 00165
 00171
 00170
 00174
12 rows selected
SQL>
```

♦

Example 5–14 shows that sometimes when a user complains of poor optimizer performance, the DBA should determine whether the strategy was chosen by the optimizer or if it was the result of complying with an outline. The strategy output for the query contains the “~S Outline outline-name used” string if an outline is used.

If your priority for a query is that the query should always succeed, then specifying a compliance level of optional is probably appropriate. If the optimizer cannot follow all of the outline directives for an optional outline, it can attempt to use other strategies to process the query.

5.9.5 Using the OPTIMIZE Clause to Choose an Outline for a Query

Oracle Rdb generates an outline and an outline ID for each query executed. The outline is generated by the optimizer, and the outline ID is based on the compilation of the entire query. When a query is compiled and the outline ID for the query is generated, Oracle Rdb looks for an outline with the same outline ID as the query. If Oracle Rdb finds an outline with the same outline ID as the query, it uses the directives in that outline to execute the query. When you define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter as "Ss", Oracle Rdb displays the outline and outline ID for each query it executes.

SQL may compile the same query slightly differently when the query is executed interactively and in a program. When this happens, the outline ID generated for the interactive query is different than the outline ID generated for the same query in a program. In this scenario, if an outline exists for the interactive query, Oracle Rdb will not automatically use that outline for the query when it is used in a program because the query and outline will have different outline IDs.

Similarly, a query may be compiled differently with different versions of Oracle Rdb due to changes made to SQL. When this happens, the outline ID generated for the query with one version of Oracle Rdb is different than the outline ID generated for the query with another version of Oracle Rdb. In this scenario, if an outline exists for the query that was created with one version, Oracle Rdb will not automatically use the outline when the query is executed using a different version because the query and outline will have different outline IDs.

Beginning with Oracle Rdb Version 6.1, you can use the OPTIMIZE USING clause of a SELECT statement to explicitly specify the outline that you want to be used with a query. This means that you can specify that a particular outline be used with a query and Oracle Rdb will attempt to use the outline you specify, even if the outline IDs for the query and the outline are different.

For example, suppose you have created an outline called WOMENS_DEGREES for the following query, which finds the degrees earned by women employees:

```
SQL> SELECT E.LAST_NAME, E.EMPLOYEE_ID, D.DEGREE, D.DEGREE_FIELD, D.YEAR_GIVEN
cont>   FROM EMPLOYEES E, DEGREES D
cont>   WHERE E.SEX = 'F' AND E.EMPLOYEE_ID = D.EMPLOYEE_ID
cont>   ORDER BY LAST_NAME
```

The following SHOW OUTLINE statement displays the outline created earlier using the outline and outline ID that Oracle Rdb generated for the query:

```
SQL> SHOW OUTLINE WOMENS_DEGREES
     WOMENS_DEGREES
 Source:

create outline WOMENS_DEGREES
id 'D3A5BC351F507FED820EB704FC3F61E8'  <-- outline id for original query
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
        join by cross to
      DEGREES 1       access path index       DEG_EMP_ID
      )
    )
  )
compliance optional     ;
SQL>
```

If you specify a query that is similar to the original query (in this example, the LIMIT TO operator is the only change), the compilation of the query may change. With the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter defined as "Ss", the output shows that Oracle Rdb generates a different outline ID for the new query:

```
SQL> SELECT E.LAST_NAME, E.EMPLOYEE_ID, D.DEGREE, D.DEGREE_FIELD, D.YEAR_GIVEN
cont>    FROM EMPLOYEES E, DEGREES D
cont>    WHERE E.SEX = 'F' AND E.EMPLOYEE_ID = D.EMPLOYEE_ID
cont>    ORDER BY LAST_NAME
cont>    LIMIT TO 10 ROWS;
   .
   .
   .
--  Rdb Generated Outline : 21-JUN-1994 15:41
create outline QO_74C62CA1A8532543_00000000
id '74C62CA1A8532543D57668C8F5BCDB92'  <--- different outline id
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
        join by cross to
      DEGREES 1       access path index       DEG_EMP_ID
      )
    )
  )
compliance optional     ;
```

```
 E.LAST_NAME      E.EMPLOYEE_ID   D.DEGREE   D.DEGREE_FIELD    D.YEAR_GIVEN
 Boyd             00244           MA         Elect. Engrg.             1982
   .
   .
   .
 Clinton          00201           MA         Applied Math              1978
10 rows selected
SQL>
```

The optimizer selects the same strategy for the new query as it did for the original query, but Oracle Rdb compiles the new query differently than the original query and generates a different outline ID.

By using the OPTIMIZE USING clause and specifying the WOMENS_DEGREES outline, you can ensure that Oracle Rdb will attempt to use the WOMENS_DEGREES outline to execute a query:

```
SQL> SELECT E.LAST_NAME, E.EMPLOYEE_ID, D.DEGREE, D.DEGREE_FIELD, D.YEAR_GIVEN
cont>    FROM EMPLOYEES E, DEGREES D
cont>    WHERE E.SEX = 'F' AND E.EMPLOYEE_ID = D.EMPLOYEE_ID
cont>    ORDER BY LAST_NAME
cont>    LIMIT TO 10 ROWS
cont>    OPTIMIZE USING WOMENS_DEGREES;
~S: Outline WOMENS_DEGREES used  <-- the query uses the WOMENS_DEGREES outline
   .
   .
   .
 E.LAST_NAME      E.EMPLOYEE_ID   D.DEGREE   D.DEGREE_FIELD    D.YEAR_GIVEN
 Boyd             00244           MA         Elect. Engrg.             1982
   .
   .
   .
 Clinton          00201           MA         Applied Math              1978
10 rows selected
SQL>
```

The output in this example shows that Oracle Rdb uses the WOMENS_DEGREES outline to execute the query.

Oracle Rdb uses the outline you specify with the OPTIMIZE USING clause unless one or more of the directives in the outline cannot be followed (for example, if the compliance level for the outline is mandatory and one of the indexes specified in the outline directives has been deleted, the outline is not used and an error message is issued).

If you specify the name of an outline that does not exist, Oracle Rdb compiles the query, ignores the outline name you specified, and searches for an existing outline with the same outline ID as the query. If it finds an outline with the same outline ID, it attempts to execute the query using the directives in that outline. If it does not find an outline with the same outline ID as the query, the optimizer selects a strategy for the query and uses that strategy for query execution.

If you specify the name of an outline that was not defined for a query, Oracle Rdb attempts to use the outline you specified and returns an error message if the outline cannot be used successfully.

You can also specify a name for a query using the OPTIMIZE AS clause. If the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as "Ss", Oracle Rdb uses the query name as the outline name in the generated outline for the query. For example:

```
SQL> ATTACH 'FILENAME mf_personnel';
   .
   .
   .
SQL> --  Specify the query name WOMENS_DEGREES for the query:
SQL> SELECT E.LAST_NAME, E.EMPLOYEE_ID, D.DEGREE, D.DEGREE_FIELD, D.YEAR_GIVEN
cont>     FROM EMPLOYEES E, DEGREES D
cont>     WHERE E.SEX = 'F' AND E.EMPLOYEE_ID = D.EMPLOYEE_ID
cont>     ORDER BY LAST_NAME
cont>     OPTIMIZE AS WOMENS_DEGREES;
~Query Name : WOMENS_DEGREES        <---- query name is displayed
   .
   .
   .
create outline WOMENS_DEGREES        <---- query name is used as outline name
   .
   .
   .
compliance optional     ;
 E.LAST_NAME      E.EMPLOYEE_ID   D.DEGREE   D.DEGREE_FIELD    D.YEAR_GIVEN
 Boyd             00244           MA         Elect. Engrg.             1982
   .
   .
   .
 Watters          00186           BA         Arts                      1975
61 rows selected
SQL>
```

See the *Oracle Rdb7 SQL Reference Manual* for more information on specifying the OPTIMIZE USING and OPTIMIZE AS clauses in select expressions.

5.9.6 Using Logical Names to Control Which Outlines the Optimizer Uses

Section 5.9.4.1 describes how to create multiple outlines for a query. When multiple outlines exist for a query, you must set the RDMS$BIND_OUTLINE_MODE logical name or the RDB_BIND_OUTLINE_MODE configuration parameter to the value of the outline mode for the outline you want the optimizer to use.

Suppose, for example, two outlines are stored for a particular query. Assume that one outline has the default outline mode value of 0 and the other outline has an outline mode value of –1. If you want the optimizer to use the outline with the outline mode value of 0 for the query, the RDMS$BIND_OUTLINE_MODE logical name or the RDB_BIND_OUTLINE_MODE configuration parameter should be set to 0 (zero). If you want the optimizer to use the other outline for the query, the RDMS$BIND_OUTLINE_MODE logical name or the RDB_BIND_OUTLINE_MODE configuration parameter should be set to –1.

If you want the optimizer to ignore any outlines that may be stored for a query, define the RDMS$BIND_OUTLINE_FLAGS logical name or the RDB_BIND_OUTLINE_FLAGS configuration parameter to the value "I". When a process has defined the RDMS$BIND_OUTLINE_FLAGS logical name or the RDB_BIND_OUTLINE_FLAGS configuration parameter as "I", the optimizer ignores any stored outlines when it processes the query.

Table 5–3 shows the values that should be set.

Table 5–3 Logical Name and Configuration Parameter Values That Specify the Outlines to Be Used for a Query

Logical Name Configuration Parameter	Value	Meaning
RDMS$BIND_OUTLINE_FLAGS RDB_BIND_OUTLINE_FLAGS	"I"	Ignore outline, if it exists.
RDMS$BIND_OUTLINE_MODE RDB_BIND_OUTLINE_MODE	valid mode	Specifies the mode of the outlines that should be chosen.

5.9.7 Visible Effects of Outlines to Users

Users are only aware that outlines are stored in a database if they:

- Use the SHOW OUTLINES statement or RMU Extract command to display or extract stored outlines.
- Have RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS set to "Ss" and notice in the strategy dump that an outline was used for a query.

- Receive an error message from Oracle Rdb when a query they issued fails because a mandatory outline for the query could not be fully complied with.

5.9.8 Invalidation of Stored Outlines

Various changes within the database (such as metadata changes) may cause an outline that is stored in the database to be unusable. For example, when a user commits the deletion of an index or table specified in an outline, the outline is marked as invalid and the invalidated outline will no longer be considered by the optimizer as a candidate outline when the query is processed.

There are two ways to determine the names of the invalid stored outlines in a database:

- SHOW OUTLINES statement

  To see whether or not a particular outline is invalid, use the SHOW OUTLINES statement and specify the name of the outline, as shown in Example 5–15.

Example 5–15 Using the SHOW OUTLINES Statement to Check the Validity of a Specific Outline

```
SQL> SHOW OUTLINES NEW_JOB_STARTS_SINCE_1982
     NEW_JOB_STARTS_SINCE_1982
*** Query outline marked as invalid ***
 Source:

create outline NEW_JOB_STARTS_SINCE_1982
id ’96231ABDD4A30FC73ABE782B14762028’
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
        join by cross to
      JOB_HISTORY 1   access path index       JOB_HISTORY_HASH
      )
    )
  )
compliance optional     ;
SQL>
```

  If you issue the SHOW OUTLINES statement without the name of a specific outline, the names of all the outlines stored in the database are displayed, but the invalid outlines are not marked as invalid.

- RMU Extract command

If you are interested in checking all of the stored outlines to determine which of them are invalid, do the following:

1. Extract all stored outlines into a file using the RMU Extract command.

   ```
   $ RMU/EXTRACT/ITEM=OUTLINES/OUTPUT=extracted-outlines.log db-name
   ```

   ```
   $ rmu -extract -item=outlines -output=extracted-outlines.log db-name
   ```

2. Using a text editor, search for the string "invalid" in the output file from the RMU Extract command. Then note the names of the invalid outlines.

You cannot revalidate a stored outline after it becomes invalid. Instead, you must delete the invalid outline using the DROP OUTLINE statement and then create a new outline. The following steps can simplify the creation of a new version of an invalidated outline:

1. Extract the invalidated outline to an output file.
2. Make the necessary changes to the extracted outline in the output file using a text editor.
3. Delete the invalidated outline using the DROP OUTLINE statement.
4. Store the new version of the outline using the output file as an SQL command procedure.

Each stored outline requires a few hundred bytes of disk space for the storage of the outline and the index or indexes used to search through the outlines efficiently.

The DBA should check all stored outlines regularly to determine whether any outlines have been marked as invalid.

5.9.9 Deleting an Outline

Example 5–16 shows how to use the SQL DROP OUTLINE statement to delete an outline.

Example 5–16 Using the SQL DROP OUTLINE Statement to Delete an Outline

```
SQL> DROP OUTLINE DEGREES_FOR_EMPS_OVER_65;
```

To delete an outline, you must have the SQL DROP privilege for every table referenced by the outline. The DROP OUTLINE statement is an online operation (other users can be attached to the database when an outline is deleted).

6

Using Oracle Rdb in a VMScluster Environment

OpenVMS VAX OpenVMS Alpha

Oracle Rdb in a VMScluster environment allows concurrent, multiple-processor database access. Oracle Rdb automatically recovers your database if a processor in your VMScluster system fails and provides optional after-image journaling to further protect the integrity of your VMScluster database.

In a properly configured VMScluster environment, Oracle Rdb can give you almost constant availability to your database.

Note

Discussions in this chapter that refer to VMScluster environments apply to both VAXcluster systems that include only VAX nodes and VMScluster systems that include at least one Alpha node, unless indicated otherwise.

This chapter describes:

- VMScluster terms and concepts important to Oracle Rdb
- How Oracle Rdb works in a VMScluster environment
- How Oracle Rdb works in a local area VMScluster environment
- How to configure your VMScluster database properly
- How to convert your single-node database to a VMScluster database
- How to monitor and maintain your VMScluster database

In addition, this chapter provides examples that:

- Create the mf_personnel database in a VMScluster environment
- Convert the mf_personnel database to a VMScluster database

Before you use your Oracle Rdb database in a VMScluster environment, you should carefully read this entire chapter. If you are not familiar with the VMScluster environment, refer to the OpenVMS documentation set for more information.

If you have questions about how to install Oracle Rdb in a VMScluster environment, see the *Oracle Rdb7 Installation and Configuration Guide*. ♦

6.1 Overview of a VMScluster Environment

OpenVMS VAX OpenVMS Alpha

This section presents a brief overview of VMScluster terms and concepts important to Oracle Rdb, but is not intended as a substitute for a full explanation of the VMScluster environment. Refer to the OpenVMS documentation set for more information about VMScluster terms and concepts.

This section includes the following topics:

- General definition of a VMScluster environment
- Methods of sharing disk file access
- Dual-ported disks
- Dual pathing between CPUs and disks
- Device-naming conventions
- Common system disk
- OpenVMS lock manager
- Distributed transactions
- Client/server computing
- Partitioned data access and shared data access ♦

6.1.1 Definition of a VMScluster Environment

OpenVMS VAX OpenVMS Alpha

A VMScluster system is a highly integrated organization of software, VAX and Alpha computers, and storage devices. With VMScluster software running on your clustered VAX and/or Alpha processors, users and applications have a single, highly integrated computing environment in which processors, storage devices, batch and print queues, and other resources are shared in the most efficient manner possible.

VMScluster systems provide a flexible way to configure computers of all sizes (desktop to data center). A VMScluster system requires the OpenVMS VAX and/or the OpenVMS Alpha operating system. VMScluster software operates in system configurations that handle cluster communications through

CI, Ethernet, DSSI, FDDI, or any combination of these link options. The OpenVMS documentation set describes the different types of VMScluster configurations. ♦

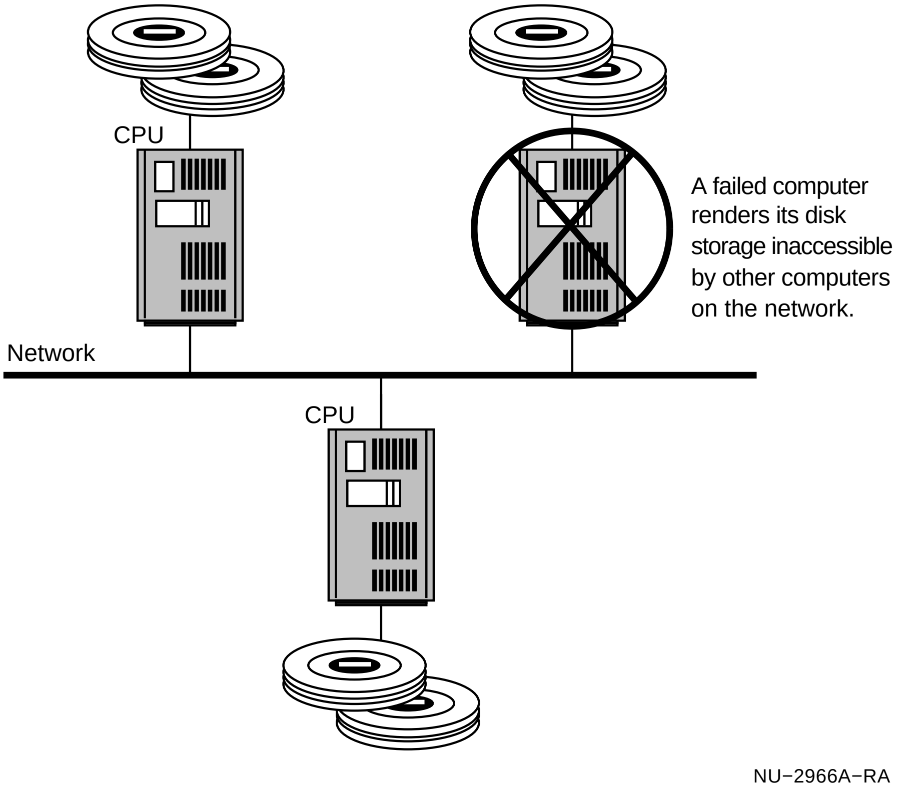
6.1.2 Shared Storage Devices

OpenVMS VAX OpenVMS Alpha

The most important feature of VMScluster software is its ability to provide transparently shared devices and files across multiple systems, which increases availability for the cluster.

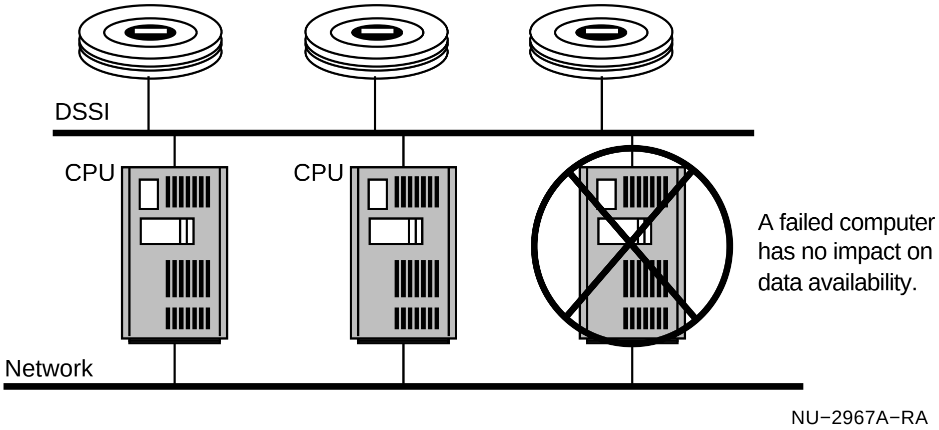
In a traditional configuration of networked systems, a single system is directly attached to its I/O devices, even though it may be networked with other systems. The result is that when a system is inaccessible, no other system on the network has access to its disks (or any other devices attached to it). A traditional configuration, such as the one shown in Figure 6–1, provides poor availability.

Figure 6–1 Traditional Configuration of Networked Systems


CPU
A failed computer renders its disk storage inaccessible by other computers on the network.
Network
CPU
NU−2966A−RA

Rather than using the traditional model of connecting I/O devices directly to systems, VMScluster configurations connect devices to communication interconnects that can be accessed by multiple systems. When one system shuts down, the remaining systems still have access to its devices. Four communication interconnects can be used by VMScluster software: CI, DSSI, Ethernet, and FDDI. The CI and DSSI are unique because they permit direct connections by disks and tape storage subsystems. For example, in a typical VMScluster configuration, a DSSI might be configured with several computer systems and several storage subsystems. Each computer system has direct access to every storage subsystem. The shutdown or failure of any system has no effect on the abilities of the other systems to access the storage. This feature, shown in Figure 6–2, results in high availability of systems and storage.

Figure 6–2 A VMScluster Configuration



In a VMScluster environment, there are two basic types of disk devices:

- Restricted access disks

  Only the node or nodes to which restricted access disks are connected can access those disks. In a VMScluster environment, OpenVMS treats each disk device as a restricted access disk, unless the disk has been mounted clusterwide, making it a cluster-accessible disk.

- Cluster-accessible disks

  Any node in the VMScluster configuration can access cluster-accessible disks.

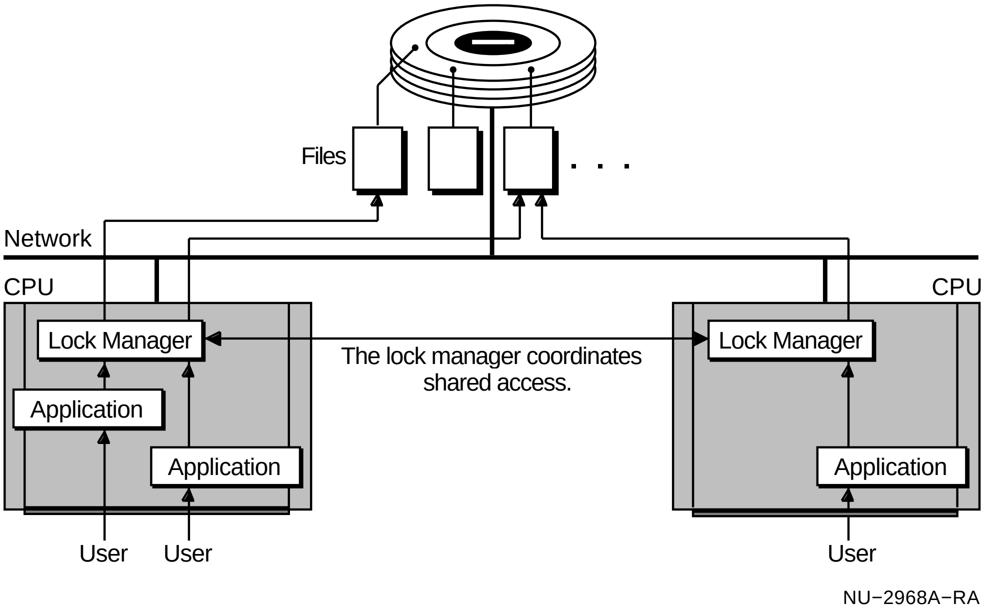
A disk is a cluster-accessible disk when the MOUNT/CLUSTER command is issued by every node in the VMScluster to mount the disk clusterwide. See the OpenVMS documentation set for more information on the MOUNT command. ♦

6.1.3 Shared Disk Files

OpenVMS VAX OpenVMS Alpha

VMScluster software also provides the ability to share files, as shown in Figure 6–3. Every system in a VMScluster can access files on any disk that is mounted clusterwide. Also, multiple systems in a VMScluster can write to a shared file simultaneously in a fully coordinated fashion. This coordination is provided by the OpenVMS lock manager. Multiple systems can even share a single system disk; multiple systems can boot off the disk and share operating system files and utilities. This feature saves disk space and greatly simplifies system management.

Figure 6–3 Disk and File Sharing



♦

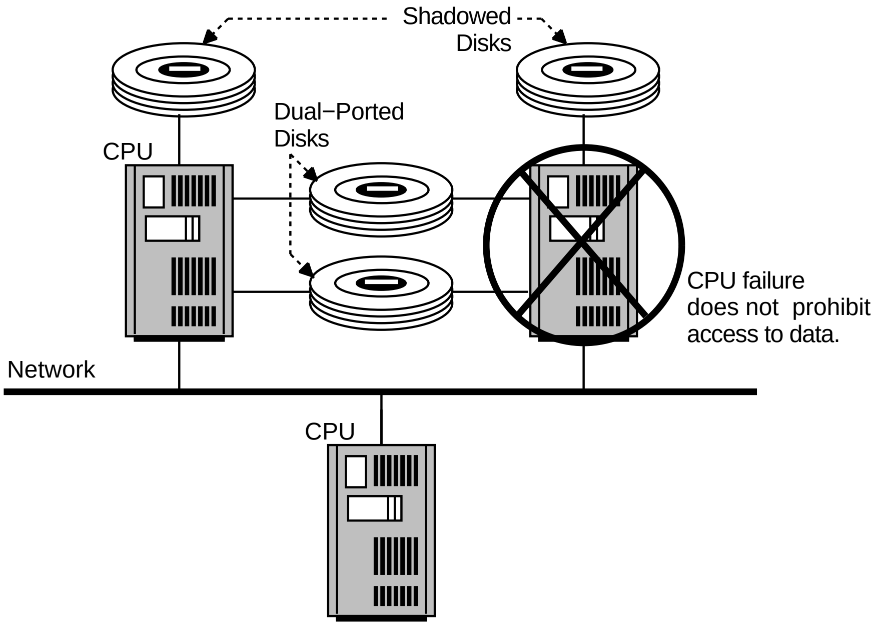
6.1.4 Dual-Ported Disks

OpenVMS VAX OpenVMS Alpha

Although the typical VMScluster system is configured with storage systems on a DSSI or CI bus, it is also possible to connect storage directly to a specific system. In the traditional network configuration, this provides lower availability due to the reliance on the host system. However, VMScluster software can enhance availability of these configurations by allowing dual porting, in which a disk drive has independent connections to two separate systems (see Figure 6–4). Then, if at least one system is available, the disk is accessible by all other systems in the VMScluster. Disks can also be shadowed in shadow sets with other disks located in the VMScluster, thereby providing enhanced availability.

The automatic recovery from system failure provided by dual porting is transparent to users and does not require any operator intervention. The OpenVMS operating system automatically switches a disk file access request from one path to another if a node through which one of the paths is established fails.

Figure 6–4 Dual-Ported Disks



NU−2969A−RA

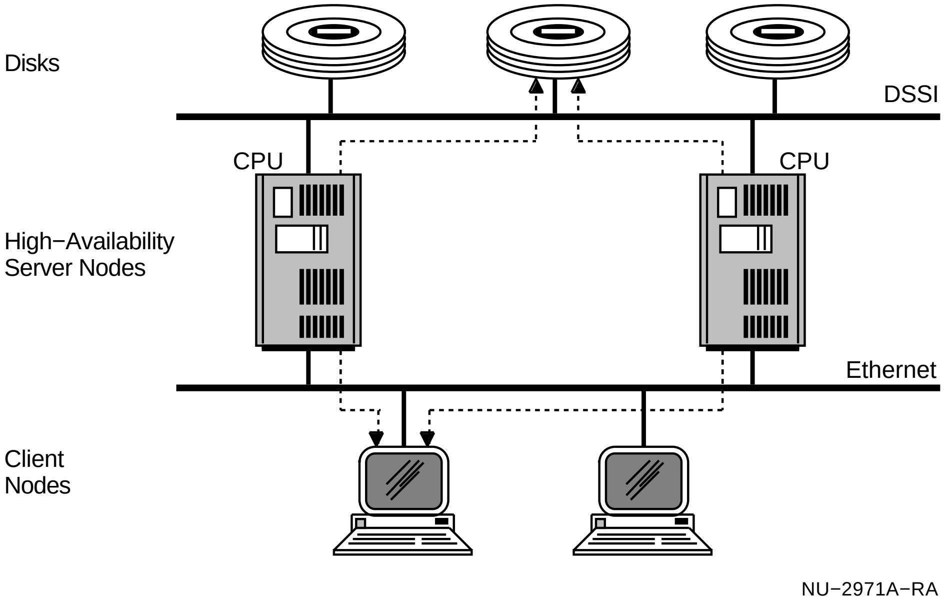
♦

6.1.5 Dual Pathing

OpenVMS VAX OpenVMS Alpha

Dual pathing is a VMScluster term for the existence of multiple paths between CPUs and disks, as shown in Figure 6–5. The dotted lines show that there is more than one path. If one path to a device fails, OpenVMS automatically fails over to an alternate path. This feature also makes it possible to build high-availability systems.

Figure 6–5 Dual-Pathed Disks



♦

6.1.6 Device-Naming Conventions

OpenVMS VAX OpenVMS Alpha

The three device-naming conventions in the OpenVMS operating system are:

- Device names

  The device-naming conventions you use in the operating system. For example:

  ```
  DUA1:
  ```

  This refers to an RA90 disk connected to your node.

- Node class device names

  The name of the node appended to the device name. For example:

  ```
  SHEMP$DUA1:
  ```

  This refers to an RA90 disk connected to node SHEMP. This node class device name allows the OpenVMS operating system to distinguish between DUA1 connected to node SHEMP and DUA1 connected to different nodes.

- Allocation class device names

A common device name for disks that are connected to two nodes. For example:

```
$2$DUA1:
```

This device name refers to an RA90 disk connected to two nodes. Each node has been assigned an allocation class identifier of 2. The OpenVMS operating system can then choose an access path to files on this disk through either node to which it is connected. If one of the nodes fails, the OpenVMS operating system can automatically switch the access to a path through the other node. Correct use of the allocation class identifier provides the advantage of the automatic failover feature of the OpenVMS operating system. You can then use the allocation class name *or* the node class name when you refer to your database files.

However, if you are using dual-ported disks, you should use a logical name that translates to the allocation class name when referring to your database files (as shown in Example 6–6). The OpenVMS operating system recognizes the allocation class name when it otherwise may not recognize the node class name. This occurs, for example, when one of the nodes to which a disk is connected is not booted. ♦

6.1.7 Common System Disk

OpenVMS VAX OpenVMS Alpha

VMScluster configurations allow multiple nodes to use the same system disk. In a VMScluster with both Alpha and VAX processors, the Alpha processors must be booted from an Alpha system disk and the VAX processors must be booted from a VAX system disk. However, VAX and Alpha processors can mount and share access to files on any disk.

If two or more processors in a cluster use a common system disk, you can also use a common SYSUAF.DAT file. A common SYSUAF.DAT file allows users to log in to any node that uses the same system disk and use the same user name, password, and default account. If you establish a common SYSUAF.DAT file for database users, make sure their default directories are on shared disk devices accessible from any node to which they can log in. Note that the default values for a number of SYSUAF process limits and quotas are higher on Alpha computers than they are on VAX computers. In general, the values in a common SYSUAF.DAT file should accommodate the largest requirements in the cluster. Then, on nodes that require smaller quotas, edit the local MODPARAMS.DAT file to adjust the system parameters to more appropriate values. See the OpenVMS system management documentation for help in determining process quotas for Alpha and VAX computers. ♦

6.1.8 OpenVMS Lock Manager

OpenVMS VAX OpenVMS Alpha

The OpenVMS lock manager provides clusterwide synchronization of resources. Oracle Rdb uses the lock manager to synchronize clusterwide updates to the root portion of the database file, to initiate the automatic recovery process when a node fails, and to coordinate concurrent updates to the same database from processes running on different nodes. ♦

6.1.9 Distributed Transactions

OpenVMS VAX OpenVMS Alpha

The OpenVMS operating system provides a set of services known as DECdtm services to facilitate transaction processing. DECdtm system services enable an application designer to implement atomic transactions that may span multiple nodes of a cluster or network. The services use a two-phase commit protocol. This support allows multiple resource managers, such as Oracle CODASYL DBMS and Oracle Rdb software, to be combined in a single transaction. See the *Oracle Rdb7 Guide to Distributed Transactions* for information on using DECdtm system services with Oracle Rdb. ♦

6.1.10 Client/Server Computing

OpenVMS VAX OpenVMS Alpha

VMSclusters can be used for client/server applications. Application designers can construct server applications that run on each node and accept requests from clients running on nodes in the VMScluster system or elsewhere in a wider network.

If the node running the application fails, the clients of that server can switch to another server running on a surviving node. The new server can access the same data on disk or tape that was being accessed by the server that failed. Also, OpenVMS volume shadowing software eliminates data unavailability in the event of a disk controller or media failure. The VMScluster is thus very available to client applications. ♦

6.1.11 Partitioned Data Access and Shared Data Access

OpenVMS VAX OpenVMS Alpha

In a VMScluster, disk storage devices can be accessed from all nodes. The application designer can choose whether the access is from one node at a time (partitioned data access) or simultaneous from multiple nodes (shared data access).

Using a partitioned data model, the application designer can construct an application that limits data access to a single node or subset of the nodes. The application runs as a server on a single node and accepts requests from other nodes in the network. Because the application runs on a single node, there is no need to synchronize data access with other nodes. Eliminating the overhead associated with synchronizing data access among nodes can improve the performance of many applications. Also, if synchronization is not required, the application designer can make the best use of buffer caches on the node and can aggregate larger amounts of data for write operations, thus minimizing I/O activity.

An application that uses partitioned data access lends itself to many types of high-performance database and transaction processing environments. VMScluster systems provide such applications with the advantage of having a storage medium that is available to all nodes even when they are accessing the data files. Thus, if the server node fails, another server running on a surviving node can assume the work and be able to access the same files. For this type of application design, VMSclusters offer the performance advantages of a partitioned data model without the problems associated with the failure of a single server.

Using a shared data model, the application designer can create an application that runs simultaneously on multiple nodes in a VMScluster, which naturally share data in a file. This type of application can prevent the bottlenecks associated with a single server and take advantage of opportunities for parallelism on multiple processors. OpenVMS RMS software can transparently share files between multiple nodes in a VMScluster. Oracle Rdb and Oracle CODASYL DBMS provide the same data sharing capability. Servers running on multiple nodes of a VMScluster system can accept requests from clients in the network and access the same files or databases. Because there are multiple servers, the application continues to function if a single server node fails. ♦

6.2 Oracle Rdb in a VMScluster Environment

OpenVMS VAX OpenVMS Alpha

Refer to the *Oracle Rdb7 Installation and Configuration Guide* for information on how to install Oracle Rdb in a VMScluster environment. Since Version 4.1 of Oracle Rdb, it has been possible to install and run multiple versions of Oracle Rdb on single nodes or one or more nodes in a VMScluster. For each node that will use a particular version of Oracle Rdb, you must install the shareable images for that Oracle Rdb version on the node.

This section compares the operation of Oracle Rdb in a VMScluster environment to the operation of Oracle Rdb in a single-node environment. ♦

6.2.1 Single-Node Environments and VMScluster Environments

OpenVMS VAX ≡ OpenVMS Alpha ≡

When you access the same database from more than one node at a time, you are using Oracle Rdb in a VMScluster environment. In this environment, Oracle Rdb establishes distributed root file access between two (or more) nodes, and establishes communications through the OpenVMS lock manager, among the monitor processes running on each node. This ensures that each node's monitor process is aware of database users on other nodes.

For the purposes of Oracle Rdb, your database can be defined as follows:

- In a single-node environment if all your database users are on a single node
- In a VMScluster environment if your database users are on more than one node at the same time ♦

6.2.2 Making a Database Accessible and Available in a VMScluster Environment

OpenVMS VAX ≡ OpenVMS Alpha ≡

When you use Oracle Rdb in a VMScluster environment, you want to make the database as accessible and available as possible. Take the following steps to make the database accessible:

1. Place all the database files (.rdb, .rda, .snp, .ruj, and .aij) on cluster-accessible disk devices (disks mounted with the MOUNT/CLUSTER command). Any node in a cluster can access cluster-accessible disks, regardless of the node to which the disks are attached. See Section 6.2.6 for more guidelines on where to place Oracle Rdb database files.
2. Make sure that the appropriate version or versions of Oracle Rdb are properly installed on each node of the VMScluster that needs to access the database. See the *Oracle Rdb7 Installation and Configuration Guide* for information on installing and starting Oracle Rdb on VMScluster nodes.
3. Specify an appropriate value with the NUMBER OF CLUSTER NODES option of the SQL CREATE DATABASE, ALTER DATABASE, or IMPORT statement. See Section 6.2.3 for information on specifying the NUMBER OF CLUSTER NODES value for a database.
4. Make Oracle CDD/Repository available (if you plan to use the Oracle CDD/Repository with the database). See Section 6.2.7.
5. If you plan to use Oracle Rdb in a local area VMScluster configuration, see the special considerations for local area VMScluster configurations in Section 6.6.1.

Take the following steps to make a database available:

1. Use dual-ported HSC disks whenever possible for database files (.rdb, .rda, .snp, .ruj, and .aij files). See Section 6.2.6 for more recommendations on device configurations for Oracle Rdb database files.
2. Place .aij files on separate disks from other database files. For example, do not place .aij files on disks that contain .rdb, .rda, .snp, or .ruj files. See Section 8.1.2.2 for more information. ♦

6.2.3 Specifying Maximum VMScluster Nodes

OpenVMS VAX OpenVMS Alpha

Use the NUMBER OF CLUSTER NODES option in the SQL CREATE DATABASE, ALTER DATABASE, and IMPORT statements to:

- Control the size of the .rdb file (1 block increase for each VMScluster node added; see the examples later in this section)
- Set the upper limit on the number of VMScluster nodes from which users can access the database

The range for the NUMBER OF CLUSTER NODES option is 1 to *n* VMScluster nodes, where *n* is the current OpenVMS software limit. If you select a value that is higher than the OpenVMS limit, it will be used to calculate an initial size for the database file, and will be displayed in the Performance Monitor General Information screen. Again, the actual limit is the maximum number of nodes permitted by the current VMScluster software based on the type of VMScluster configuration.

A database created with the NUMBER OF CLUSTER NODES IS 17 option will be larger than the same database created with the NUMBER OF CLUSTER NODES IS 10 option. For each node in a VMScluster environment from which Oracle Rdb users might access a shared Oracle Rdb database, additional data structures are required in the .rdb file.

If you omit the NUMBER OF CLUSTER NODES option from the SQL CREATE DATABASE, ALTER DATABASE, and IMPORT statement, the default remains 16. You can use this option on the IMPORT statement to set a new value for a database created with an earlier version of Oracle Rdb.

If you use the NUMBER OF CLUSTER NODES option with the SQL CREATE DATABASE and ALTER DATABASE statements, a subsequent SQL EXPORT and IMPORT operation will retain this value as it was in the .rbr file following the SQL EXPORT statement because it saves the parameter's value in the .rbr file.

If you attempt to access a database from a VMScluster node and, in doing so, exceed the maximum nodes parameter, an error occurs as shown in Example 6–1.

Example 6–1 Exceeding the Maximum Nodes Parameter

```
SQL> -- VMScluster max nodes value is 5,
SQL> -- and database is already at the limit:
SQL> ATTACH 'FILENAME $222$DUA12:[TOP]mf_personnel';
%RDB-F-SYS_REQUEST, error from system services request
-RDMS-F-NORTUPB, no more user slots are available in the database
SQL>
```

Example 6–2 creates a single-file personnel database and defines a value of 20 for the number of VMScluster nodes using the NUMBER OF CLUSTER NODES IS option. This change from a default of 16 to 20 VMScluster nodes would be made if the database was to reside on a shared disk in a 20-node local area VMScluster configuration. Example 6–3 shows a Performance Monitor General Information screen that displays the new value of the node count.

Example 6–2 Specifying the NUMBER OF CLUSTER NODES Value

```
SQL> CREATE DATABASE FILENAME personnel NUMBER OF CLUSTER NODES IS 20;
SQL> EXIT
```

Example 6–3 Displaying the Maximum NUMBER OF CLUSTER NODES Value

```
Node: TRIXIE         Oracle Rdb V7.0-00 Performance Monitor 24-JUN-1996 14:15:32
Rate: 3.00 Seconds            General Information           Elapsed: 00:00:16.86
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]PERSONNEL.RDB;1         Mode: Online
--------------------------------------------------------------------------------

Database created at 24-JUN-1996 14:14:46.13
Maximum user count is 50
Maximum node count is 20      <------------ Notice
Database open mode is Automatic
Database close mode is Automatic
Snapshot mode is Automatic
Statistics collection is enabled
Default recovery-unit journal filename is "Not Specified"
Date of last backup is 17-NOV-1858 00:00:00.00
Fast incremental backup is enabled




--------------------------------------------------------------------------------
Exit Help Menu Options Refresh Set_rate Write !
```

Example 6–4 uses the NUMBER OF CLUSTER NODES option to the SQL IMPORT statement to decrease the node count parameter to a value of 15. For multifile databases, you can use the SQL ALTER DATABASE statement and specify a new value in the NUMBER OF CLUSTER NODES option.

Example 6–4 Changing the NUMBER OF CLUSTER NODES Value

```
$ SQL
SQL> EXPORT DATABASE FILENAME personnel.rdb INTO personnel_test.rbr
cont> WITH EXTENSIONS;

SQL> IMPORT DATABASE FROM personnel_test.rbr
cont> FILENAME personnel_test.rdb
cont> NUMBER OF CLUSTER NODES 15;     <---- Decrease to 15.

Exported by Oracle Rdb V7.0-00 Import/Export utility
A component of SQL V7.0-00
Previous name was personnel.rdb
It was logically exported on 28-MAY-1996 15:29
   .
   .
   .
```

(continued on next page)

Example 6–4 (Cont.) Changing the NUMBER OF CLUSTER NODES Value

```
Database NUMBER OF CLUSTER NODES was 20, now is 15
   .
   .
   .
```

♦

6.2.4 Multiple Monitor Processes

OpenVMS VAX OpenVMS Alpha

In a single-node environment, Oracle Rdb uses a detached monitor process to control database recovery, coordinate database open and close operations, and maintain a list of active database users.

Because VMScluster configurations do not allow processes to be shared among nodes, Oracle Rdb uses a database monitor process on each processor in the VMScluster system from which an Oracle Rdb database can be accessed. The monitor processes communicate with each other through the OpenVMS lock manager. A monitor process on one node in a VMScluster environment is aware of database users on other nodes.

You must include the line @SYS$STARTUP:RMONSTART in the system startup file for each processor that will access an Oracle Rdb database. Because each version of Oracle Rdb has a monitor process, in a multiversion environment there is a separate RMONSTART.COM file for each version of Oracle Rdb installed. In the multiversion environment, therefore, you start the monitor for a particular version of Oracle Rdb by specifying the appropriate RMONSTART.COM file for that version of Oracle Rdb (for example, @SYS$STARTUP:RMONSTART61 to start the monitor for Oracle Rdb V6.1). See the *Oracle Rdb7 Installation and Configuration Guide* for more information on starting Oracle Rdb using RMONSTART.COM.

When a processor fails in a VMScluster environment, the OpenVMS lock manager alerts Oracle Rdb monitor processes on other nodes where users of the same database exist. One of these monitor processes then initiates the Oracle Rdb recovery procedure to roll back the transactions that were in progress for users on the failed node. Refer to Section 6.5 for further information on the Oracle Rdb database recovery procedure. ♦

6.2.5 Partitioned Lock Trees

OpenVMS Alpha

By default, the active locks for all the resources in an Oracle Rdb database belong to a single resource tree rooted under the database lock for that database. In a VMScluster environment, all the locks and resources in a database's resource tree are mastered by OpenVMS on one of the nodes in the VMScluster.

OpenVMS chooses the VMScluster node on which all the locks and resources are mastered; this choice can be influenced by the usage of the various locks. Because only one node can master a database resource tree at a time, the other nodes in the VMScluster have to exchange messages with the mastering node for every lock request. When a single node handles all the messages related to lock requests in a VMScluster environment with high transaction rates, a performance bottleneck can result.

If your database is an Oracle Rdb for OpenVMS Alpha database, you can alleviate the performance bottleneck caused by a single resource tree per database by using the LOCK PARTITIONING IS ENABLED clause of the SQL CREATE or ALTER DATABASE statement, as shown in Example 6–5.

Example 6–5 Enabling Lock Partitioning

```
SQL> CREATE DATABASE FILENAME pers_test
cont>       LOCK PARTITIONING IS ENABLED;
```

When partitioned lock trees are enabled for a database, there is no longer one resource tree per database that includes all the resources in the database. Instead, area locks are separated from the database resource tree and are independently mastered on the VMScluster node that has the highest traffic for that resource. OpenVMS determines the node that is using each resource the most and moves the resource hierarchy (area lock and underlying page and record locks) to that node.

The performance gain from enabling partitioned lock trees is especially significant for partitioned applications that use the database. For example, if you have a partitioned application in which the VMScluster node NODE1 is the only node accessing storage area AREA1, NODE1 does not have to handle any lock requests from other VMScluster nodes for AREA1. Because NODE1 does not have to handle lock requests from other nodes for AREA1, the application runs faster when partitioned lock trees are enabled.

You can disable partitioned lock trees by using the LOCK PARTITIONING IS DISABLED clause of the ALTER or CREATE DATABASE statement.

For more information about the SQL syntax, see the CREATE DATABASE statement and the ALTER DATABASE statement in the *Oracle Rdb7 SQL Reference Manual*. ♦

6.2.6 Deciding Where to Place Oracle Rdb Files in a VMScluster Environment

OpenVMS VAX OpenVMS Alpha

This section provides recommendations to help you decide where to place your Oracle Rdb files in a VMScluster environment. The goals of these recommendations are to:

- Provide access to database files from appropriate nodes in a VMScluster environment
- Allow automatic database recovery in the event of a node failure
- Ensure virtually uninterrupted availability of your database

For the Oracle Rdb recovery process to work correctly in a VMScluster environment, all Oracle Rdb database and journal files must be placed on disks that are always accessible to all the nodes in the cluster using the database. If you do not place all your database and journal files on cluster-accessible devices, Oracle Rdb may not be able to initiate and complete the database recovery procedure when a node fails. If Oracle Rdb is not able to recover the database, all access to the database is suspended until the recovery is completed.

After Oracle Rdb recovers the database, users on a failed node can log in to another node and continue using the database. Therefore, you should ensure that users have accounts on surviving nodes or that your VMScluster uses a common SYSUAF.DAT file. See Section 6.5 for further information on the transaction rollback.

You can choose among various options for VMSclusters that provide you with high performance access and with high availability:

- Dual-ported HSC disks

  Provide you with both high performance and an available alternate path to your files, should one of the HSC subsystems fail.

- Single-ported HSC disks

  Provide you with high performance, but do not automatically provide an alternate path to your files.

- Dual-ported served disks

  A served disk is a locally connected disk that has been explicitly set up as cluster-accessible using the MSCP server. See the OpenVMS system management documentation for more information on served disks.

A dual-ported served disk automatically provides an alternate path to your files, should one VAX or Alpha processor fail. However, served disks provide high performance only for the processors to which they are connected.

- Single-ported served disks

  Can be used to make your Oracle Rdb files available to all processors in the VMScluster environment. However, they provide high performance only for the processor to which they are connected, and do not provide an alternate path to your files.

To ensure continuous database access by surviving nodes when a node in the VMScluster configuration fails, do not place any Oracle Rdb files on disk devices that are cluster-mounted on single-ported, served disks. If you place any Oracle Rdb files on single-ported disks, and the node to which the disks are connected fails, the Oracle Rdb recovery procedure cannot access those files. Activity on the database ceases until the failed node can be restarted.

To reduce the chances of losing access to your database due to node failure, consider the following facts:

- If you place your files on dual-pathed HSC disks, you lose access to your database only when both HSC subsystems fail at the same time.
- If you place your files on dual-pathed served disks, you lose access to your database only when both processors fail at the same time.
- If you place your files on a single-pathed HSC disk, you lose access to your database when that HSC subsystem fails.
- If you place your files on a single-pathed disk that is not an HSC subsystem, you lose access to your database when that VAX or Alpha processor fails.

When you use dual-pathed disks, always use a logical name for the allocation class name whenever you refer to your database files. See Section 6.1.6 for information on allocation class names for disks and Example 6–6 for information on using logical names for allocation class names.

Note

> In a local area VMScluster configuration, if one of the boot nodes fails, cluster operations are suspended until the node rejoins the cluster. This condition is normal and ensures the integrity of shared cluster resources.

The following guidelines can help you decide where to place the database files:

- Database root (.rdb), storage area (.rda), and snapshot (.snp) files

  In a single-node environment, Oracle Rdb maps the root portion of the database file, or root header, as a global section of memory. Because VMScluster configurations do not allow memory to be shared between nodes, Oracle Rdb in a VMScluster environment maintains copies of the root header in a page-file section on each node that uses a database.

  The root file itself remains on the disk where you created it. Therefore, this disk must be accessible from all nodes in the VMScluster system that will access the database. If the root file of a database is not available to a node, that database cannot be used by that node.

  The OpenVMS lock manager ensures that all root file copies are identical.

  The root, storage area, and snapshot files must be accessible from every node that intends to access the database. Oracle Rdb must be able to create and maintain VMScluster distributed root file access.

- Recovery-unit journal (.ruj) files

  Oracle Rdb must be able to complete its automatic recovery procedure should a node fail. Recovery can complete only if *all* .ruj files are accessible from every node that accesses the database. To ensure that all .ruj files are accessible, follow these rules:

  - Define the RDMS$RUJ logical name in the system table to refer to a common directory.
  - Define the RDMS$RUJ logical name for each process to refer to a common directory.
  - If you permit .ruj files to reside on the user's default directory, that directory's device must be on a cluster-accessible disk.

- After-image journal (.aij) files

  The .aij files must be accessible from every node that accesses the database. This ensures that all processes that access the database will be able to write to the .aij files.

  To learn about reducing I/O contention in multiuser database access, read Section 3.2 and Section 8.1.1, which describe disk I/O operations and the corresponding effects on database performance. Placing the .aij files on different devices than the database files is one way to minimize I/O contention on the database disks.

Note that all the recommendations made in this section are based on the assumption that you want the database to be accessible from several or all of the nodes in your VMScluster, that you want it to be a high-availability database, and that you want Oracle Rdb to automatically recover the database in the event of a node failure. For some databases, high availability might not be as important as security. For example, if your database is a private database or a sensitive database on a public system, you may be more concerned about database security than about having the database available from other nodes in a VMScluster. In this case, you could put the database files on disks on your local node that are not mounted clusterwide, which will prevent users from other VMScluster nodes from accessing these files. Performance should also be better if the database files are on the local node, because your database requests will not have to travel over the network to access the database files and because locks will be mastered on the local node. However, you sacrifice database availability because if your node or one of the disks containing database files fails, you will not be able to access the database until the problem with your local node is resolved. When the local node fails, Oracle Rdb will recover the database automatically when the node reboots. ♦

6.2.7 Oracle CDD/Repository Requirements

OpenVMS VAX ≡ OpenVMS Alpha ≡

Your repository definitions must be available to all processes on all nodes that need them, if you have Oracle CDD/Repository, the data repository, installed. The types of processes that access the repository include:

- Procedures, command files, and interactive user operations that execute data definition statements that affect the repository
- Programs that require the repository for database definitions

Before you create an Oracle Rdb database in a VMScluster environment, you must ensure that your main repository files are on a device accessible to all users.

In general, you should always place your main repository and your compatibility repository on a shareable cluster disk that is always accessible to users from any node in the VMScluster configuration. Placing these files on a shareable cluster disk and establishing a single logical repository has the following advantages:

- It allows users in your organization access to the repository even if their particular system fails.

- It enables your organization to share data definitions across the entire cluster. A single repository limits redundant data storage and helps ensure consistency among all database entity definitions. If you maintain separate repositories on different systems, you risk discrepancies in data definition and inconsistencies in updating data.

The following three commands define a logical name for the shareable disk on which the repository will reside, establish a single logical repository, and place the repository files on a shareable VMScluster disk.

```
$ DEFINE/SYSTEM CDD_DISK $2$DUA11:
$ DEFINE/SYSTEM/EXECUTIVE_MODE CDD$COMPATIBILITY CDD_DISK:[CDD]
$ DEFINE/SYSTEM CDD$DICTIONARY CDD_DISK:[DMU]
```

These commands place your main dictionaries on an HSC disk called $2$DUA11.

These logical names, as well as others necessary to access Oracle CDD/Repository repositories, are defined by the procedure CDDSTRTUP.COM, which should be invoked by each SYSTARTUP_V5.COM or SYSTARTUP_VMS.COM that starts up a node that will use these main repository files.

You may place your main repository files on a restricted-access disk. However, you must ensure that all processes that require data definitions are able to access them at that location. ♦

6.3 Creating the mf_personnel Database in a VMScluster Environment

OpenVMS VAX ≡ OpenVMS Alpha ≡

This section presents an example that shows how you can distribute the mf_personnel database in a VMScluster environment to allow access from any node. The necessary database files for this example include the following:

cdd.dic
mf_personnel.rdb
empids_low.rda
empids_mid.rda
empids_over.rda
salary_history.rda
departments.rda
emp_info.rda
jobs.rda
resume_lists.rda
resumes.rda
empids_low.snp
empids_mid.snp
empids_over.snp
salary_history.snp
departments.snp
emp_info.snp
jobs.snp
resume_lists.snp
resumes.snp
mf_personnel.aij (optional)

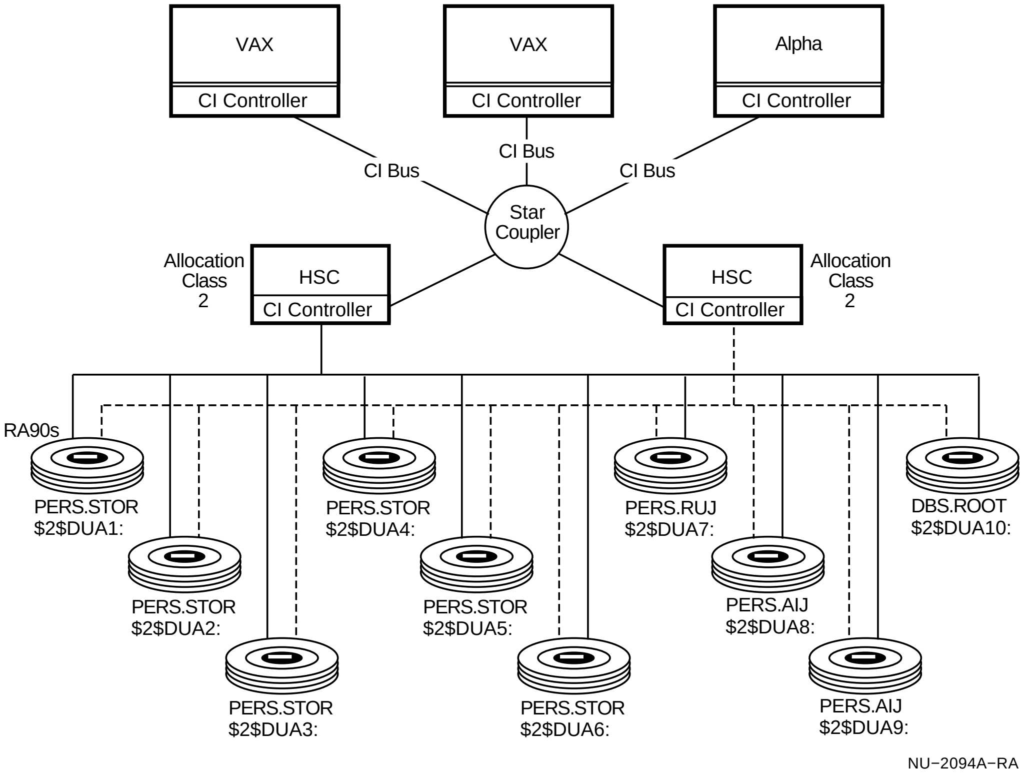
The configuration includes the following hardware:

- Ten RA90 disks, dual-pathed to two HSC subsystems
- Three VAX processors

- Additional VMScluster hardware requirements include CI buses, CI controllers, and a star coupler

The SQL CREATE DATABASE statement and its qualifiers place these files on HSC disks that have assigned allocation class names. Figure 6–6 shows the disks that contain the database files in a VMScluster environment.

Figure 6–6 Sample Placement of Database Files



Because the VMScluster manager assigned the number 2 as the allocation class identifier for each HSC subsystem, the allocation class names for the 10 RA90 disks are $2$DUA1, $2$DUA2, $2$DUA3, $2$DUA4, $2$DUA5, $2$DUA6, $2$DUA7, $2$DUA8, $2$DUA9, and $2$DUA10.

In Example 6–6, logical names are defined for the disks referred to in Figure 6–6.

Example 6–6 Defining Logical Names for the Disks Used for the Database Files

```
$ DEFINE/SYSTEM DB_DISK $2$DUA10:
$ DEFINE/SYSTEM DISK1 $2$DUA1:
$ DEFINE/SYSTEM DISK2 $2$DUA2:
$ DEFINE/SYSTEM DISK3 $2$DUA3:
$ DEFINE/SYSTEM DISK4 $2$DUA4:
$ DEFINE/SYSTEM DISK5 $2$DUA5:
$ DEFINE/SYSTEM DISK6 $2$DUA6:
$ DEFINE/SYSTEM RUJ_DISK $2$DUA7:
$ DEFINE/SYSTEM AIJ_DISK1 $2$DUA8:
$ DEFINE/SYSTEM AIJ_DISK2 $2$DUA9:
```

In Example 6–7, the directories for the database files are defined.

Example 6–7 Creating Directories for the Database Files

```
$ CREATE/DIRECTORY DB_DISK:[DBS.ROOT]
$ CREATE/DIRECTORY DISK1:[PERS.STOR]
$ CREATE/DIRECTORY DISK2:[PERS.STOR]
$ CREATE/DIRECTORY DISK3:[PERS.STOR]
$ CREATE/DIRECTORY DISK4:[PERS.STOR]
$ CREATE/DIRECTORY DISK5:[PERS.STOR]
$ CREATE/DIRECTORY DISK6:[PERS.STOR]
$ CREATE/DIRECTORY RUJ_DISK:[PERS.RUJ]
$ CREATE/DIRECTORY AIJ_DISK1:[PERS.AIJ]
$ CREATE/DIRECTORY AIJ_DISK2:[PERS.AIJ]
```

You can define the RDMS$RUJ logical name for each process to refer to a cluster-accessible device that contains all the .ruj files. For example, the command in Example 6–8 defines the RDMS$RUJ logical name to refer to the RUJ_DISK:[PERS.RUJ] directory, which contains .ruj files for all processes that access the database.

Example 6–8 Defining the RDMS$RUJ Logical Name for the Directory for the .ruj Files

```
$ DEFINE/SYSTEM RDMS$RUJ RUJ_DISK:[PERS.RUJ]
```

If you plan to use a systemwide directory for your .ruj files, include this command in your SYSTARTUP_V5.COM or SYSTARTUP_VMS.COM file. Define this directory on a device other than the database files. You can also use the default, SYS$LOGIN, to place .ruj files per process in each user's default directory. If you do not define the logical name, RDMS$RUJ, to refer to a common directory, make sure that all database users log in to accounts whose default directories are on cluster-accessible disks.

Example 6–9 shows the use of the SQL CREATE DATABASE statement using the defined logical names.

Example 6–9 Defining the Sample Database

```
SQL> CREATE DATABASE FILENAME 'DB_DISK:[DBS.ROOT]mf_personnel'
cont> PATHNAME 'SYS$COMMON[HEADQUARTERS]mf_personnel'
cont> NUMBER OF USERS IS       50
cont> NUMBER OF BUFFERS IS     20
cont> NUMBER OF CLUSTER NODES IS 16
cont> NUMBER OF RECOVERY BUFFERS IS 20
cont> BUFFER SIZE IS            6 BLOCKS
cont> SNAPSHOT IS ENABLED
cont> DICTIONARY IS REQUIRED;
cont> DEFINE STORAGE AREA DEPARTMENTS
cont>   FILENAME DISK3:[PERS.STOR]departments.rda
cont> ALLOCATION IS 25 PAGES
cont> PAGE FORMAT IS MIXED
cont> SNAPSHOT FILENAME DISK4:[PERS.STOR]departments.snp
cont> SNAPSHOT ALLOCATION IS 10 PAGES
cont> END DEPARTMENTS STORAGE AREA;
SQL>
```

The statements in Example 6–9 create the database files, mf_personnel.rdb, departments.rda, and departments.snp on a dual-pathed HSC disk device and directory using the directory names DB_DISK:[DBS.ROOT], DISK3:[PERS.STOR], and DISK4:[PERS.STOR]. The mf_personnel database can be accessed by any node in the cluster.

The SQL ALTER DATABASE statement names the .aij files and their location, as shown in Example 6–10.

Example 6–10 Specifying the Locations of the .aij Files

```
SQL> ALTER DATABASE 'DB_DISK:[DBS.ROOT]mf_personnel'
cont> ADD JOURNAL AIJ_ONE
cont> FILENAME 'AIJ_DISK1:[PERS.AIJ]aij1.aij'
cont> ADD JOURNAL AIJ_TWO
cont> FILENAME 'AIJ_DISK2:[PERS.AIJ]aij2.aij';
SQL>
```

If a node in the VMScluster system fails, a monitor process on a surviving node can access the database, .ruj, and .aij files necessary to recover the database.

Because this VMScluster configuration uses a common system disk with a common SYSUAF.DAT file, if one of the nodes fails, users can log in to one of the surviving nodes and continue to use the database.

If one HSC subsystem fails, access to the database files automatically switches to the path through the other HSC.

Use this list of steps to guide you through the process of creating your Oracle Rdb database in a VMScluster environment:

1. Decide which nodes require access to the database. You should ensure that:

   a. Oracle Rdb can be run on each node

   b. The Oracle Rdb shareable images are installed on each node

   c. The @SYS$STARTUP:RMONSTART.COM procedure is in the SYSTARTUP_V5.COM or SYSTARTUP_VMS.COM file each node uses

   d. The logical name RDMS$RUJ is defined for each node that requires database access

   e. All user accounts log in to default directories on cluster-accessible devices (where appropriate)

2. Determine which disks are accessible from the user nodes that you determined needed access. You should choose the best paths and devices options when:

   a. High availability is your priority

   b. High performance is your priority

3. Decide whether or not your users can log in to another node if the node they normally use fails. In the event of node failure, you must ensure one of the following conditions is true:

   a. Users have accounts on the other node or nodes.

   b. All nodes share a common SYSUAF.DAT file. You may want to use different SYSUAF.DAT files for Alpha processors and VAX processors to allow for the different memory management working set quotas (SYSUAF process limits and quotas are higher on Alpha processors).

4. Decide where to place the .ruj files by either:

   a. Defining a system logical to refer to a single directory

   b. Creating your database users' default directories on cluster-accessible disks

   c. Defining process logicals to refer to a user directory located on one of the cluster-accessible disks

5. Decide where to place the database file. The database file must be accessible from all nodes using the database.

6. Issue the SQL CREATE DATABASE and ALTER DATABASE statements to create the database on the specified devices and to create an .aij file on a device other than where the .rdb file resides. ♦

6.4 Converting a Single-Node Database to a VMScluster Database

OpenVMS VAX OpenVMS Alpha

This section describes how to convert a single-node database to a database that runs efficiently in a VMScluster environment. The steps listed in Section 6.3 that describe how to create a multifile database also apply in this case except for the last step. You can make your current, single-node database available to users in the following ways:

- Use the RMU Backup and RMU Restore commands.

  Refer to the *Oracle Rdb7 Guide to Database Maintenance* for information on using the RMU Backup and RMU Restore commands for this procedure. This method is preferred because it is the fastest. You will need to specify the Directory, Root, File, and Snapshots qualifiers with the RMU Restore command. The *Oracle RMU Reference Manual* describes the syntax and qualifiers for this command.

- Alternatively, use the SQL EXPORT and IMPORT statements.

  Example 6–11 shows the use of the SQL EXPORT and IMPORT statements to convert a single-node database to a database that will be accessible in a VMScluster. This method usually takes longer to execute than an RMU Backup and RMU Restore operation and therefore is not the preferred method to accomplish this particular task. Refer to the *Oracle Rdb7 Guide to Database Maintenance* for additional information on applications that use the SQL EXPORT and IMPORT statements. See the *Oracle Rdb7 SQL Reference Manual* for information on syntax and arguments for these two statements.

Section 3.2 and Section 8.1.1 provide information that can help you improve database performance by distributing I/O operations across several storage devices. Section 6.2.6 describes placement of Oracle Rdb files.

With the SQL EXPORT and IMPORT statements, you can take the following actions:

- Issue an SQL EXPORT statement to create an interchange file of your database.
- Issue an SQL IMPORT statement to distribute your database and journal files on shareable disk devices.

Converting your database from a single node to a VMScluster environment is not difficult. However, you must carefully evaluate the alternatives available to you before you begin this task.

The logical names DB_DISK, DISK3, DISK4, AIJ_DISK1, and AIJ_DISK2 in Example 6–11 are defined in Example 6–6. Use these logical names to place the mf_personnel database files on the appropriate disk devices. Example 6–11 shows how to use the SQL EXPORT and IMPORT statements to convert a single-node database to a VMScluster database.

Example 6–11 Using the SQL EXPORT and IMPORT Statements to Convert a Single-Node Database to a VMScluster Database

```
$ SQL
SQL> EXPORT DATABASE FILENAME DB_DISK:[DBS.ROOT]mf_personnel
cont> INTO DISK11:[DBS.EXP]mf_personnel.rbr WITH EXTENSIONS;
SQL>
SQL> IMPORT DATABASE FROM DISK11:[DBS.EXP]mf_personnel.rbr
cont> FILENAME DB_DISK:[DBS.ROOT]mf_personnel.rdb
cont> PATHNAME "SYS$COMMON:[HEADQUARTERS]mf_personnel"
cont> NUMBER OF USERS IS        50
cont> NUMBER OF BUFFERS IS      20
cont> NUMBER OF CLUSTER NODES IS 16
cont> NUMBER OF RECOVERY BUFFERS IS 20
cont> BUFFER SIZE IS             6 BLOCKS
cont> SNAPSHOT IS ENABLED
cont> DICTIONARY IS REQUIRED
cont>
cont> CREATE STORAGE AREA DEPARTMENTS
cont>   FILENAME DISK3:[PERS.STOR]departments.rda
```

(continued on next page)

Example 6–11 (Cont.) Using the SQL EXPORT and IMPORT Statements to Convert a Single-Node Database to a VMScluster Database

```
cont> ALLOCATION IS 25 PAGES
cont> PAGE FORMAT IS MIXED
cont> SNAPSHOT FILENAME DISK4:[PERS.STOR]departments.snp
cont> SNAPSHOT ALLOCATION IS 10 PAGES
cont> END IMPORT;
SQL> FINISH;
SQL>
SQL> ALTER DATABASE 'DB_DISK:[DBS.ROOT]mf_personnel'
cont> ADD JOURNAL AIJ_ONE
cont> FILENAME 'AIJ_DISK1:[PERS.AIJ]aij1.aij'
cont> ADD JOURNAL AIJ_TWO
cont> FILENAME 'AIJ_DISK2:[PERS.AIJ]aij2.aij';
SQL>
```

The following summary describes the steps necessary to convert your existing, single-node database to a cluster-accessible database:

1. Issue an SQL EXPORT statement to create an interchange file of your database.
2. Decide where to place the database files.

   Make sure all Oracle Rdb files, including the .rdb, .rda, .snp, .ruj, and .aij files, are located on devices accessible from all nodes that require access to the database.
3. Issue an SQL IMPORT statement to restore your database and distribute the database and journal files on cluster-accessible disk devices. Define database parameters again using the SQL IMPORT statement clauses.
4. Test your new database to ensure that data definitions are available from both the database itself and the repository, and that the database and its data are accessible from each node.
5. Delete the old database. ♦

6.5 Automatic Recovery Procedure

OpenVMS VAX OpenVMS Alpha

When a node in a VMScluster configuration fails, Oracle Rdb performs an automatic recovery of the database to bring the database to a consistent state.

The normal operating state of an Oracle Rdb database in a VMScluster environment includes an Oracle Rdb monitor process that runs on each node from which a database can be accessed. When a node fails, all processes on it are aborted. If any of the aborted processes were accessing a database, the OpenVMS lock manager grants a special lock (called a deadman) to a monitor on one of the surviving nodes from which one or more users is accessing the database. When a monitor on a surviving node is granted a deadman lock to a database that was accessed from a failed node, that surviving node suspends all activity in the database for all the nodes in the VMScluster. The surviving node that was granted the deadman lock then starts a database recovery process (DBR process).

The DBR process brings the database to a consistent state. The steps in the database recovery are different depending on whether or not fast commit processing is enabled for the database. When a node fails and a database that was being accessed on the failed node has fast commit processing enabled, the DBR process first uses the .aij file to recover all the committed transactions in the database since each user's last checkpoint. This recovery of committed transactions is known as a redo operation. Then the DBR process rolls back the uncommitted transactions recorded in the .ruj files. This rollback of uncommitted transactions is known as an undo operation.

When a node fails and a database that was being accessed on the failed node does not have fast commit processing enabled, the DBR process rolls back the uncommitted transactions recorded in the .ruj file, but no redo operation is necessary because the committed updates have already been written to the database area files.

If a monitor process on a surviving node cannot access all the required .ruj files for a database, the database is shut down. To complete recovery, the database must be recovered from a node that has access to the .ruj files.

After the database has been recovered, users from the failed node can log in to a surviving node in the VMScluster configuration and resume using the database (if these users have accounts on the surviving nodes).

This scenario illustrates the importance of creating and maintaining your database and journal files on shareable disk devices.

Note that when more than one version of Oracle Rdb is installed on a VMScluster, there will be a version-specific monitor process on each node for each version of Oracle Rdb that is installed. In an Oracle Rdb multiversion environment, each node has more than one monitor process. If a node fails in a multiversion environment, the recovery procedure occurs for each version of Oracle Rdb that is installed.

If a database is accessed only by the node that fails, the recovery procedure is the same as for any single-node database. The recovery process will be initiated the next time a process attaches to the database. The failure of an HSC subsystem is treated differently by Oracle Rdb than the failure of a processor. No recovery procedure is initiated because no user processes are aborted. The effect on your database depends on the configuration of your VMScluster environment:

- If your HSC disks are dual-pathed, the OpenVMS operating system automatically switches your database access path through an alternate HSC subsystem. Access to your database is not interrupted.
- If your HSC disks are single-pathed, and any of your database files are on an HSC disk, database access is discontinued until the HSC subsystem is again operational. The next time a process attaches to the database, a recovery procedure will be started.

Section 6.5.1 and Section 6.5.2 describe the Performance Monitor database recovery screens. ♦

6.5.1 Performance Monitor Recovery Statistics Screen

OpenVMS VAX OpenVMS Alpha

The Performance Monitor provides the Recovery Statistics screen, which identifies various recovery phases and shows information on how long each phase took to complete.

The Recovery Statistics screen is useful for identifying an excessive number of abnormal process failures. In addition, the screen is useful for determining the proper database attribute and parameter settings to maximize run-time performance and minimize recovery downtime.

Note that the Recovery Statistics screen provides global information on all failed process recoveries, not on individual process recoveries.

You access the Recovery Statistics screen from the Journaling Information submenu. The following example shows a Recovery Statistics screen:

```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor  3-FEB-1996 08:04:27
Rate: 1.00 Second            Recovery Statistics          Elapsed: 00:26:45.95
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....
process attaches             0        0        0.0            4           1.0
process failures             0        0        0.0            1           0.2
DB freeze len x100           0        0        0.4          684         171.0
Tx REDO count                0        0        0.0            1           0.2
  redo time x100             0        0        0.2          389          97.2
Tx UNDO count                0        0        0.0            1           0.2
  undo time x100             0        0        0.0            9           2.2
No UNDO needed               0        0        0.0            0           0.0
Tx committed                 0        0        0.0            0           0.0
Tx rolled back               0        0        0.0            0           0.0
No resolve needed            0        0        0.0            1           0.2
AIJ recover x100             0        0        0.0            6           1.5
GB recover x100              0        0        0.0            0           0.0
Cache recover x100           0        0        0.0            0           0.0
RUJ file reads               0        0        0.0            1           0.2
AIJ file reads               0        0        0.0            9           2.2
--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

See the Performance Monitor help for information about this screen. ♦

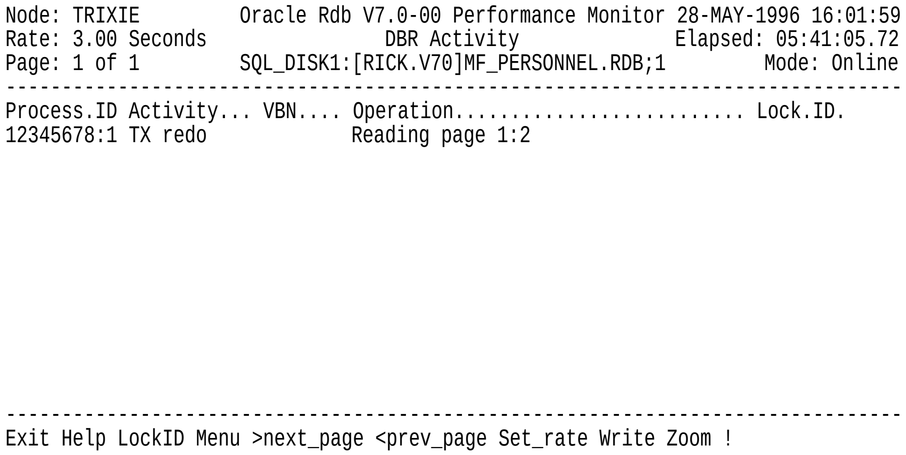
6.5.2 Performance Monitor DBR Activity Screen

OpenVMS VAX ☰ OpenVMS Alpha ☰

The Performance Monitor provides the DBR Activity screen, which allows you to obtain information on active DBR processes on the current node. The DBR Activity screen provides one line of information for each DBR process that is active on the node from which the Performance Monitor is invoked. Note that if there are no active DBR processes on the node, the screen is empty.

You access the DBR Activity screen from the Process Information submenu.

If this screen is not used, the only method available to users to determine if the DBR process is running is to use the RMU Show Users utility. However, the RMU Show Users utility indicates only that the DBR process is running; it does not indicate what type of progress DBR is making in the recovery operation. The following example shows a DBR Activity screen:



```
Node: TRIXIE        Oracle Rdb V7.0-00 Performance Monitor 28-MAY-1996 16:01:59
Rate: 3.00 Seconds                DBR Activity             Elapsed: 05:41:05.72
Page: 1 of 1        SQL_DISK1:[RICK.V70]MF_PERSONNEL.RDB;1        Mode: Online
--------------------------------------------------------------------------------
Process.ID Activity... VBN.... Operation.......................... Lock.ID.
12345678:1 TX redo             Reading page 1:2








--------------------------------------------------------------------------------
Exit Help LockID Menu >next_page <prev_page Set_rate Write Zoom !
```

See the Performance Monitor help for information about this screen. ♦

6.6 Maintaining and Monitoring Your Database

OpenVMS VAX OpenVMS Alpha

This section describes the normal operation and maintenance of an Oracle Rdb database in a VMScluster environment.

Most operations performed on Oracle Rdb databases in a VMScluster environment are the same as those performed on Oracle Rdb databases in a single-node environment. ♦

6.6.1 Local Area VMScluster Configuration Considerations

OpenVMS VAX OpenVMS Alpha

A typical local area VMScluster configuration consists of a large processor and several small processors. You must ensure that each smaller processor in your configuration has enough memory to run your applications. Because creating databases requires more memory than other applications, you may have to dedicate one processor to the task of database creation. You may want to consider using the local processors in your local area VMScluster system for paging and swapping files.

Monitor the following AUTOGEN parameters closely:

- PAGEDYN
- NPAGEDYN
- GBLSECTIONS
- GBLPAGES

Make sure the values of each of these parameters are large enough to accommodate the number of interactive users and anticipated batch job load, if any, for your application.

The boot node or nodes in a local area VMScluster environment incur a performance penalty because they must act as disk servers for the satellite nodes. A **satellite** node is any processor that is started up remotely from a boot node's system disk. Generally, these nodes consume the cluster's resources, although they can be set up to provide disk serving and batch processing resources.

If a boot node in a local area VMScluster fails, cluster operations are suspended until the node rejoins the cluster. This condition is normal and ensures the integrity of shared cluster resources. ♦

6.6.2 Monitoring Your Database

OpenVMS VAX OpenVMS Alpha

You can display information about the database from any node in the cluster. The RMU Show Users and RMU Show System commands display information about the users on the node from which the command is issued.

To display information about the database, use:

- RMU Analyze Areas
- RMU Analyze Lareas
- RMU Analyze Indexes
- RMU Analyze Placement

To display information about users on your node, use:

- RMU Show Users
- RMU Show System

These commands execute only for a single node at a time. This restriction is due to the distributed root file access and multiple monitor processes required to implement Oracle Rdb in a VMScluster environment. You will have to execute the RMU Show Users and RMU Show System commands on each node in the VMScluster system on which database user processes reside, to obtain a full picture of activity for that database. You can do this by placing these commands into command procedures. These procedures can then be submitted simultaneously to batch queues on each node from which users have access to a database. Use the SQL SET OUTPUT full-file-spec statement to direct the output to disk files in a single directory. You can then display or print the contents of the files.

You can use the RMU Dump Users command to display information for all the database user processes for a cluster:

```
$ RMU/DUMP/USERS mf_personnel
```

You can also use the Performance Monitor to monitor database activity. See Section 2.2 for details on monitoring your database. ♦

7

Tuning Concepts and Methodology

System tuning is not a new concept in time-sharing environments. A well-tuned computer system biases resource distribution towards the types of workloads for which the machine is intended. This chapter explores how system tuning can affect database performance. Although this chapter uses a manufacturing environment as a model, the database tuning concepts that are developed apply to all database environments.

In computer-integrated manufacturing (CIM) environments, the ability to quickly store and access information is critical to the process requirements found on the shop floor. In addition, manufacturing plants cannot always justify more powerful hardware to improve response time. In this situation, tuning the system for optimal database performance affects the whole operation and provides a low-cost solution to the problem.

7.1 What Is Tuning?

Tuning is adjusting system or database parameters to obtain an overall optimum resource usage for the target workloads without adding additional central processing unit (CPU), memory, or input/output (I/O) capacity. Note that adding more memory or another disk drive to the environment is *not* tuning; tuning enables you to manage with the available resources. The trick is to figure out what workloads are using the resources and which of these workloads are most important. You then set up the system so that the most important workloads always have resources available when they need them.

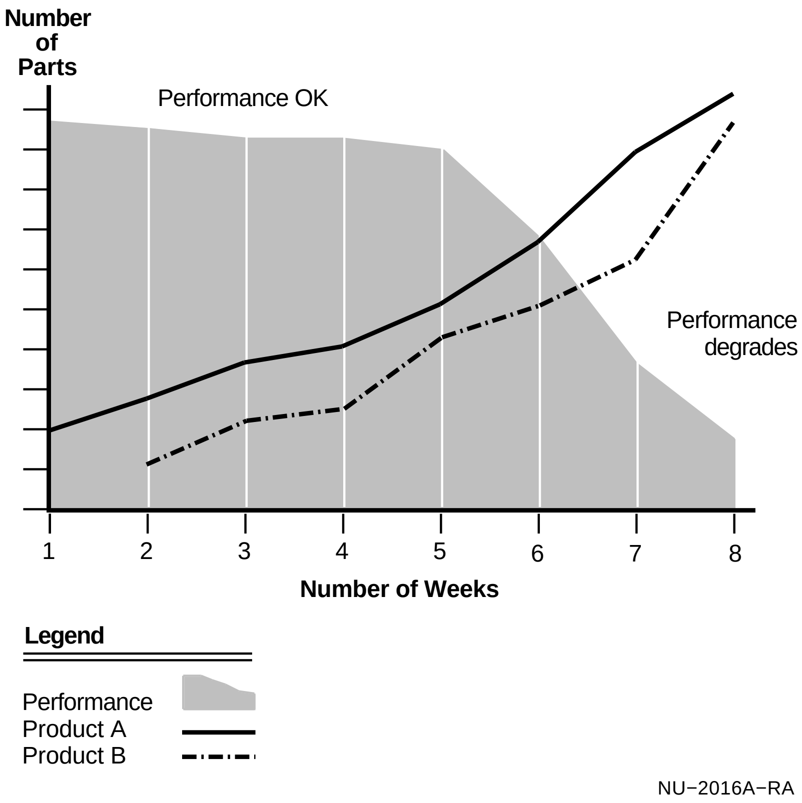
7.2 Determining When to Tune

Tuning is not normally an action that needs to occur continuously within your environment. Instead, system monitoring should occur regularly. Tuning normally takes place when response time degrades or when system monitoring shows a resource shortage.

You must understand your workload and resource usage to determine when tuning is necessary. If your application is less responsive than expected, the need for a critical resource may be causing the problem. Additional research is necessary.

If the workload has changed over time, you should again consider tuning. This situation is common in a manufacturing environment where volume fluctuations or product ramping may cause the workloads to change gradually. System and database parameters that worked fine at 10 percent product volumes may not work as well for higher percentages. When performance degradation becomes noticeable, investigate. Figure 7–1 emphasizes this situation.

Figure 7–1 Tuning Because of Change in Workloads



Increased hardware capacity is the most common reason for you to tune your environment. For example, if you double the memory capacity of your machine, the working sets established for your old hardware configuration may no longer be optimal. Adjusting the working sets to take advantage of the additional memory may improve performance dramatically for those applications that have large memory requirements. Similarly, if you add another disk to the configuration, modifying the database schema to spread the data over the additional device could help eliminate an I/O or contention problem. Tuning enables you to use the new capacity to its fullest advantage.

7.3 Types of Resources

Oracle Rdb uses two types of resources, exclusive and shared.

Fortunately, Oracle Rdb uses few resources that must be used *exclusively* by the workload that requires it. When an exclusive resource must be used, a less important workload may prevent an important workload from accessing the resource and thus delay critical work. This may degrade performance. For example, when an index is defined for a table, the table (the resource) must be used exclusively by the definer. This increases contention for workloads that need access to the table.

The other type of resource is the *shared* resource. This is the more common resource found in the database environment. The usual problem involving this resource type is that the resource becomes over-shared. To illustrate this problem, let an apple represent a resource in your environment. If you slice the apple into several pieces and give those pieces to people (who represent the workloads), the slice may not be enough to satisfy their hunger. They may have to wait for another slice. If some workloads are more important than others, you will want to tune the computer system to ensure that those more important workloads immediately receive the percentage of the resource that they require.

Tuning is a complex process, with many factors that can affect the outcome of the proposed environment change. Section 7.5 offers a database tuning methodology to help you to deal with this complexity.

7.4 Sample Database Application

This chapter uses examples to illustrate tuning concepts. Many of these examples are based on a manufacturing resource planning (MRP II) database application named PRODUCT_DB. As you read the examples, imagine that you designed the PRODUCT_DB application.

An MRP II application is a system designed to manage and execute a variety of business functions that apply to manufacturing planning and control of material and capacity resources. MRP II is a transaction-driven system, and it is highly dependent on record accuracy. It also requires strict discipline from its users, who place heavy demands on the system for response and flexibility.

A typical MRP II application, such as PRODUCT_DB, contains information about a manufacturing plant. It includes the information (bill of materials) needed to track the different subcomponents of an assembled unit, as well as the different parametric data generated during testing to ensure a quality product. The applications that run on the database are used by advanced development engineers to refine the manufacturing process, by quality engineers to ensure that a good product is being built, and by manufacturing to track information on the shop floor.

Table 7–1 lists and briefly describes each table in the PRODUCT_DB database.

Table 7–1 PRODUCT_DB Database Tables

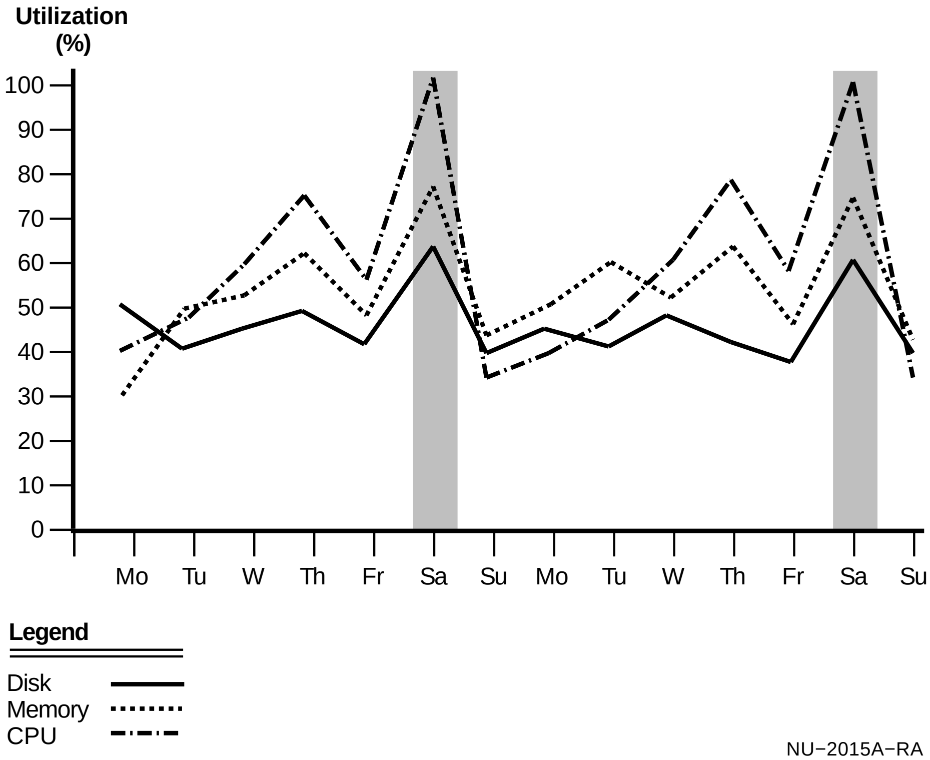
Table Name	Description
LOT_INFORMATION	Tracks material through the shop floor.
SHIPPING_INFORMATION	Tracks when and where the completed product has been shipped.
PARTS	Links a part to a given lot name.
PART_DEFECT	Identifies why a given part was defective.
DEFECT_DESCRIPTIONS	Translates the defect codes into a description that can be sent back to the vendor identifying why the part was rejected.
LOT_DATA	Stores parametric test data associated with a lot.
PROCESS_SPECIFICATIONS	Controls the test limits for the product at a given process step.

7.5 Tuning Methodology

This section describes a database tuning methodology. It provides a six-step sequence that you can use to approach any database tuning problem. Frequently, when a potential problem is identified, the cause of the problem is unknown and the information concerning the problem is vague. Take the following scenario, for example:

Users performing data entry on a MRP II application notice that performance has degraded over a period of months. They complained to the system manager, who called in the database expert (you). Where do you start?

Figure 7–2 Analyze Resource Use



NU−2015A−RA

1. Understand the workloads running in the environment.

   The first step is to understand the workloads in your environment. Does an application that generally has a 2-second response now suddenly have a 5-second response? Has the environment changed recently? (For example, has a new version of the software or a layered product been installed?) Does the problem occur only at certain hours of the day or on certain days of the week, as shown in Figure 7–2?

It is useful to have performance reports available for study during this phase of the process. Tools such as the OpenVMS Monitor utility (MONITOR) can be used to collect this information. Begin by looking daily at groups of system resources to understand when minimum and maximum resource utilization occurs. For example, in a manufacturing environment, resource utilization might slowly increase during the week and peak on the day that shipment quotas are due. Understanding this, you would not attempt to diagnose a performance problem on a Monday, when utilization is at a minimum. Instead, you would schedule your tuning analysis for a Saturday, when resources are more strained and your analysis is likely to be more effective. (Limiting resources are more evident during peak periods.) You might also notice increased utilization at the end-of-quarter ship deadlines and the end-of-year ship deadlines.

Transaction and volume analysis information (transaction volumes, type of transaction, length of transaction) is useful if it is available. The more you know about the applications running in the environment, the more insight you will have into the cause of the problem.

2. Check for hardware problems.

   The users who are experiencing the problem may not be aware of, or understand, the computing environment in which they are working. Before you undertake any tuning, check to ensure that no major hardware problems exist on the system. If you are running in a cluster environment, check to see that the normal number of machines is running. (If a machine is off line, it might account for the user's perception that response is slow.)

   OpenVMS VAX OpenVMS Alpha

   Use the SHOW ERROR command from DIGITAL Command Language (DCL) to check for problems that could be affecting your environment or workloads. For example, are there many errors on the database disk?

   The SHOW ERROR command will not indicate all hardware problems, however. For example, if a large number of database disks are spread across several HSC subsystems and one HSC fails, then all database disks might switch to the same HSC. This switch to a single HSC subsystem would slow things down. One way to diagnose this problem is to use the DCL SHOW DEVICES command to determine which HSC subsystem the disks are on. (If the disk is on an HSC, the HSC name appears in parentheses after the device name.) ♦

   It is also a good idea to check with the system manager to ensure that the hardware configuration has not changed recently. It is possible that a faulty piece of hardware was removed from the system until a replacement can be found. A user accustomed to the additional capacity of the original configuration would notice the change as a response time problem.

3. Focus on the problem area.

   Carefully examine I/O, memory, CPU, and lock utilization trends to focus on the problem area. To ensure that you do not make conclusions from data that does not reflect the normal operation of the workloads in the environment, you should try to review several weeks of information. If several weeks of data are not available, use what you have. The risk involved with using smaller samples is that you may base your conclusions on an off-peak period when the cause of the performance degradation is not exhibited. Note that any resource that is more than 65 percent busy may start to degrade performance. Your goal is to spread the work to other less utilized resources that have the capacity for it.

4. Determine the cause of the problem.

   Once you have categorized the problem as being in one of the four basic areas (I/O, memory, CPU, or locks), you can look in depth at system level parameters, database parameters, or at the application itself to determine the options that are available to remove the bottleneck.

5. Choose a viable solution and execute the change.

   This step involves selecting the best available solution and applying the change to the environment. Issues that you should consider when you choose a solution include cost, time to implement, risk, and potential for improvement.

6. Monitor the results.

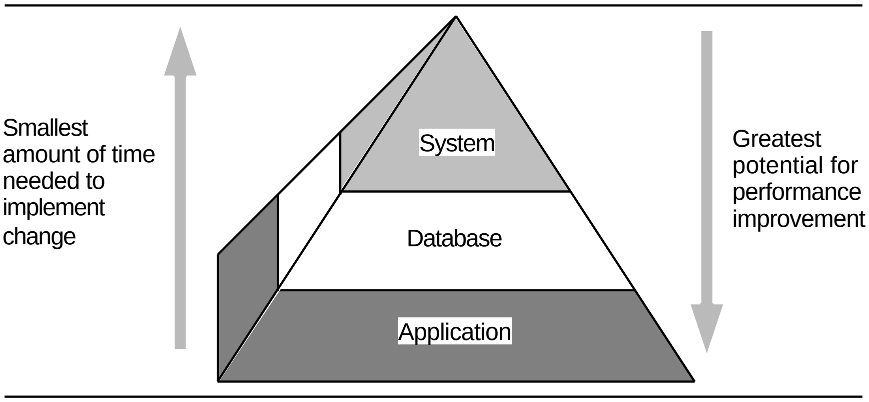
   After a change has been made, the environment should be monitored to ensure that the situation improves.

7.6 Determining What to Tune

Before you explore the tuning methodology in greater detail, you need to determine what to tune. Section 7.5 noted that you should focus on four basic areas of tuning resources: I/O, memory, CPU, and locks. The potential for performance degradation exists any time that utilization of any one of these resources gets above 65 percent.

Once you have selected an area to focus on, you must consider several factors before you decide on a tuning solution. Figure 7–3 illustrates this concept.

Figure 7–3 Areas of Potential Improvement


Smallest amount of time needed to implement change
System
Database
Application
Greatest potential for performance improvement

NU−2021A−RA

7.6.1 Tuning the System

Figure 7–3 shows a breakdown by environmental component. The three components that you can concentrate on to improve performance are **system, database,** and **application.** This breakdown should help you address such issues as risk and cost.

OpenVMS VAX OpenVMS Alpha The system tuning component consists of the parameters found in the OpenVMS System Generator utility (SYSGEN), the OpenVMS Authorize utility (AUTHORIZE), and other dynamic parameters like RDM$BIND_BUFFERS. ♦

If you look at the pyramid in Figure 7–3, you will notice that the system has the smallest area. This shows that system tuning has the smallest potential for performance gain.

Although the potential performance improvement for tuning at the system level may be smaller than the other areas, beginning a tuning session at this level offers some advantages. First of all, the time required to implement a change at this level is small. A number of the affected parameters are dynamic, and can be modified right away. Other parameters take effect during the next login or the next time the system is rebooted. Another advantage is that the risk is smaller when a change can be undone almost immediately. In a manufacturing environment, where system downtime can directly affect product volumes, low-risk tuning changes that improve performance without affecting production during their implementation phase are valuable enhancements. If the changes required the plant to shut down, they might not be looked upon as favorably.

Cost is another advantage of tuning at the system level. Expensive programmer and database administrator resources are usually not required to make changes at this level. The problems encountered here are typically more global than a single application, so internal understanding of the application code is not required. This leads to a final point. Tuning at the system level is less complex because less application-specific information is necessary.

7.6.2 Tuning the Database

The database tuning component contains the database definition language, which performs such functions as specifying file placement, number and size of database buffers, and initial file allocation. This component contains all the nonsystem parameters that can be used to solve a problem without requiring modification to the application code. The database area in Figure 7–3 falls between the system and application areas. This shows that tuning at the database level can have a larger effect than system tuning but has less potential effect than modifying the application.

Modifications to the parameters at the database level are in general more complex than at the system level. More explicit knowledge of the data and how it is being used is required. Normally database tuning must be coordinated and implemented through a database administrator. This can lead to longer delays than a change at the system level.

Few of the database parameters can be changed dynamically. Most changes require exclusive access to the database. Others require the database to be exported and then imported for the change to have maximum effect. The drawback of this is higher risk and cost. If the environment in which the database operates requires 7X24 uptime, then tuning at the database level has an unavoidable impact. The cost is higher because the additional expense of the idle plant must be factored in. The risk is higher because there may be unexpected delays in getting the database back on line. Development environments are sometimes set up to reduce the risk of making the changes on line. All proposed database changes are tested in the development database before they are implemented in the production environment. Unfortunately, because most development databases are not the same size as the production database, the risk is not removed altogether. When trying to decide whether to use a development database, issues such as the cost of setting up the development database, the cost of the required hardware, and the cost of making all changes twice must be weighed against the cost of an unexpected problem that keeps the database off line longer.

7.6.3 Tuning the Application

The application tuning component contains the application code, the transaction information, and the interaction with the user. The application area, the biggest of the three areas in Figure 7–3, is at the base of the pyramid. This illustrates that tuning at the application level can have the largest potential impact on overall performance.

Modifications at the application level are the most complex, as they require detailed knowledge of the application code, the types and sizes of the transactions, and the environment in which the application is working. In a large environment, this can involve multiple applications and programmers. Changes at the application level often take much longer to implement than at the other levels.

Typical changes at the application level require modification and restructuring of application code. The possibility exists that new bugs may be introduced when application code is modified. This has all sorts of ramifications on risk and cost. The same argument for a test environment that was discussed in Section 7.6.2 can be made here. The next paragraph gives an example of a problem that would be resolved at this level.

Assume that you have determined that the user's response time problem only occurred during certain periods of the day and that it appeared to be caused by excessive locking at the PARTS table. Using the Performance Monitor, you have determined from the Stall Messages screen that a large percentage of the users are waiting for a database page to be freed. To determine the root cause of the problem, you trace the process IDs of the applications that are running, and find one of the running applications sitting at a Modify Part display, with the operator on a break. A quick look at the application code shows that the application was written so that user input is required from the middle of a write transaction. One way to avoid the locking problem is to recode the application to get the required information from the user before the transaction is started. This keeps the actual write transaction to the database as short as possible, which reduces contention.

8 Diagnosing a Database Resource Bottleneck

A database application is normally I/O-intensive, so this chapter suggests that you analyze resources in the following order:

- I/O
- Memory
- CPU
- Locking

Each decision tree in this chapter provides an organized approach that you can use to isolate, identify, analyze, and solve a particular performance problem. Figure 8–1, for example, shows one approach to analyzing an I/O resource bottleneck. As in all decision trees in this manual, solutions appear in boxes with a heavy border. *Go to* boxes indicate that you should proceed to a figure depicting a decision tree with that label. The remaining boxes are transitional; they ask specific questions intended to guide you to a solution. Other decision trees follow these same conventions. Each figure is intended to help you to recommend tuning solutions that can be easily implemented with low risk. Wherever possible, the decision trees are set up so that low-risk solutions are reached before other more difficult options are considered.

Note

As you read this chapter, note that the SQL term *table* is used in the text, although the RMU output in this chapter and elsewhere displays the equivalent term *relation*.

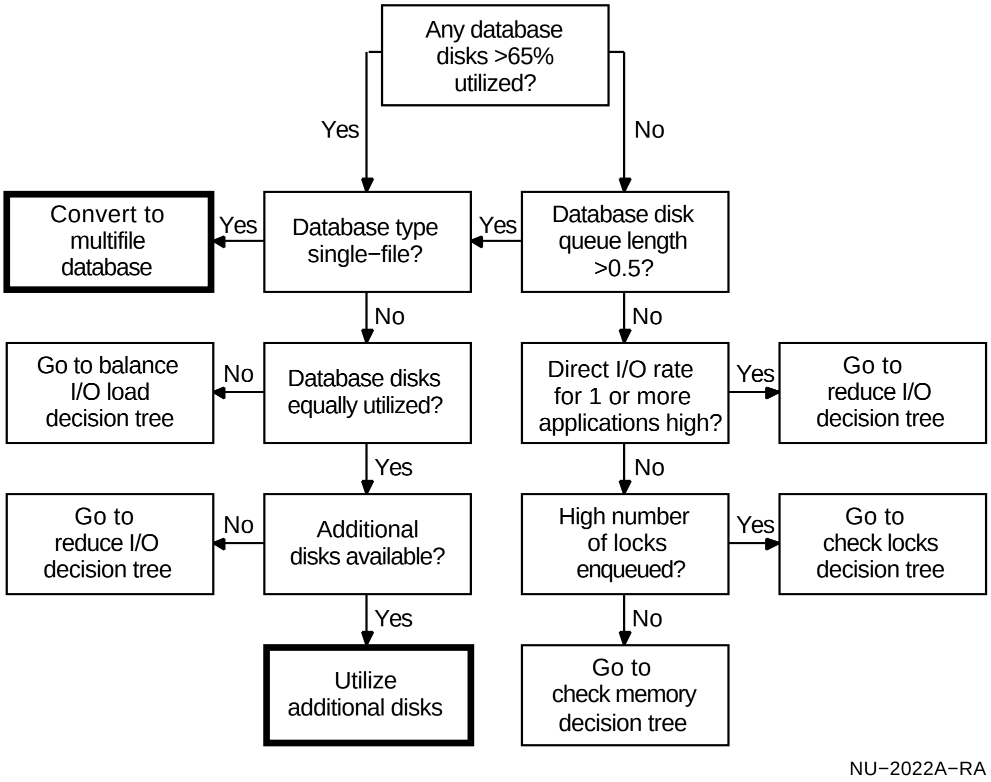
8.1 Analyzing I/O Resources

When you review I/O resources, you should concentrate on detecting I/O resource bottlenecks, balancing I/O load, and reducing I/O operations. Section 8.1.1, Section 8.1.2, and Section 8.1.3 provide details of how you can perform these tasks.

8.1.1 Detecting I/O Resource Bottlenecks

As Figure 8–1 shows, the best place to start investigating your system for a potential I/O resource bottleneck is at the disk devices. If the utilization is over 65 percent, performance is likely to be affected to some extent.

Figure 8–1 Decision Tree: Check for an I/O Resource Bottleneck



Any database disks >65% utilized?
Yes
No
Convert to multifile database
Yes
Database type single−file?
Yes
Database disk queue length >0.5?
No
No
Go to balance I/O load decision tree
No
Database disks equally utilized?
Direct I/O rate for 1 or more applications high?
Yes
Go to reduce I/O decision tree
Yes
No
Go to reduce I/O decision tree
No
Additional disks available?
High number of locks enqueued?
Yes
Go to check locks decision tree
Yes
No
Utilize additional disks
Go to check memory decision tree
NU−2022A−RA

If you have a single-file database and performance is a bigger concern than reduced database complexity, your first step is probably to change it to a multifile database. You can determine the approximate maximum throughput for a transaction in a single-file database to see whether it is worthwhile to change the database to a multifile database. The formula for determining the maximum throughput for a given transaction in a single-file database is shown in Example 8–1.

Example 8–1 Formula for Determining the Throughput Possible for a Transaction in a Single-File Database

```
                                   Disk I/Os Performed per Second
  Transactions per Second    =   ---------------------------------
                                               N  +  L
```

To determine the maximum possible throughput for a transaction in your single-file database, you first need to know approximately how many I/Os per second can be performed by the database disks. Example 8–2 assumes the disks can perform 30 I/Os per second. Second, you need to know the number of I/Os that must be performed by the transaction (a value of 7 in our example for variable N) and the number of I/Os to write to the .rdb, .ruj, and .aij files (a value of 3 in our example for variable L) for the transaction. So, for our example, the maximum possible throughput for the transaction is 3 transactions per seconds (tps), as shown in Example 8–2.

Example 8–2 Determining the Throughput Possible for a Given Transaction in a Single-File Database

```
                                      30
  3 Transactions per Second   = ---------------
                                   7  +  3
```

If 7 I/O operations are performed by the transaction, the best throughput you can expect in a single-file database is 3 transactions per second (tps). If you want to improve the throughput, you should change the database to a multifile database. Note that just making the database multifile does not yield large performance gains. Rather, the multifile option gives you the *potential* to explore more options during the tuning process. If a bottleneck develops in a multifile database, you still must analyze and understand where the bottleneck is, and make the appropriate changes to remove it.

If a database is a multifile database, one tuning option is to spread it across multiple disks to reduce I/O contention on the single device. If you find that one of your database disks is still highly utilized, you have several options. If additional disks are available, spread the database out more to take advantage of them. If this is not an option, then you need to improve the balance of the I/O load across the existing disk resources. When you spread or rebalance your database, remember that the database root and .aij files can become database hot spots in a high transaction environment. The root file hot spot can be alleviated by using fast commit transaction processing and commit to journal optimization, if applicable, as described in Section 4.1.5 and Section 4.1.5.3. Also, the root file and .aij file should each be on its own disk. If the root file cannot be placed on its own disk, place it on a disk that is not highly utilized. If you are operating in an environment that has different types of disk devices, you should place the root file and .aij file on the fastest devices.

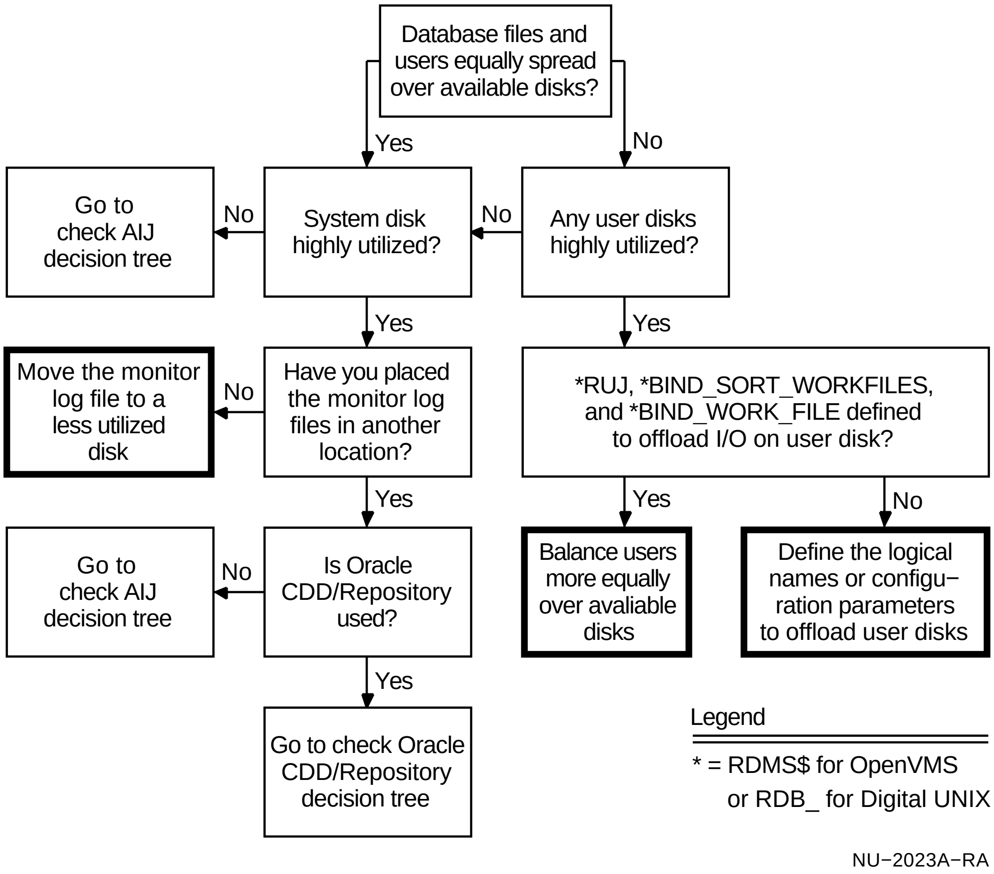
If the disk devices do not seem to be a problem, another area to check for I/O resource bottlenecks is the application. If an application has a high average direct I/O rate, it is possible that performance is being affected by the disk that services the requests. Because disk requests are relatively slow compared to memory requests, high direct I/O rates for an application are a potential area for tuning.

If careful review establishes that you do not have an I/O resource problem, then the next area to examine is memory. See Section 8.2 for information about how to review memory resources.

8.1.2 Balancing I/O Load

Once you have a multifile database, check to see that it is spread equally across the available disks, which can help reduce database hot spots. Also, if the default directories for database users are spread equally across the user disks, this can help prevent contention problems. Figure 8–2 summarizes the steps in balancing I/O load. If the system disk is highly utilized, moving the default Oracle Rdb monitor log file location to another disk may help to reduce contention.

Figure 8–2 Decision Tree: Balance I/O Load


Database files and users equally spread over available disks?
Yes
No
Go to check AIJ decision tree
No
System disk highly utilized?
No
Any user disks highly utilized?
Yes
Yes
Move the monitor log file to a less utilized disk
No
Have you placed the monitor log files in another location?
*RUJ, *BIND_SORT_WORKFILES, and *BIND_WORK_FILE defined to offload I/O on user disk?
Yes
Yes
No
Go to check AIJ decision tree
No
Is Oracle CDD/Repository used?
Balance users more equally over avaliable disks
Define the logical names or configu− ration parameters to offload user disks
Yes
Go to check Oracle CDD/Repository decision tree
Legend
* = RDMS$ for OpenVMS or RDB_ for Digital UNIX
NU−2023A−RA

If the users' disks are highly utilized, you can define the logical names RDMS$BIND_SORT_WORKFILES, RDMS$RUJ, and RDMS$BIND_WORK_FILE or the configuration parameters RDB_BIND_SORT_WORKFILES, RDB_RUJ, and RDB_BIND_WORK_FILE to help reduce contention. These logical names and configuration parameters deal with default placement of Oracle Rdb temporary files.

OpenVMS VAX OpenVMS Alpha

On OpenVMS, the RDMS$BIND_SORT_WORKFILES logical name defines the number of temporary files used by the OpenVMS Sort utility (SORT). Oracle Rdb uses SORT for all sorting operations that the optimizer decides are too large to be done in memory, for example, large SORTED BY queries and large index builds. RDMS$BIND_SORT_WORKFILES has a maximum value of 10 (files). The default value is 2.

While RDMS$BIND_SORT_WORKFILES controls the number of sort work files that are used, SORTWORK0 through SORTWORK9 control the placement of the sort work files. Distributing these files over a number of disks reduces contention during the sort operation. ♦

You can also define RDMS$BIND_SORT_WORKFILES or RDB_BIND_SORT_WORKFILES and its related sort work files for each application for finer granularity. For detailed information, refer to Section A.89.

The RDMS$RUJ logical name and the RDB_RUJ configuration parameter control the default placement of the .ruj file. Defining this name is another way to reduce contention on a user disk. The following example shows how to redirect the .ruj file on OpenVMS:

```
$ DEFINE RDMS$RUJ RUJ$DISK:[OFFLOADED_RUJS]
```

Finally, you can use the RDMS$BIND_WORK_FILE logical name or the RDB_BIND_WORK_FILE configuration parameter to redirect temporary files created by Oracle Rdb to another location. You can also use this logical name or configuration parameter to reduce contention on the user disk. The following example shows how to redirect the location where these temporary files are created on OpenVMS:

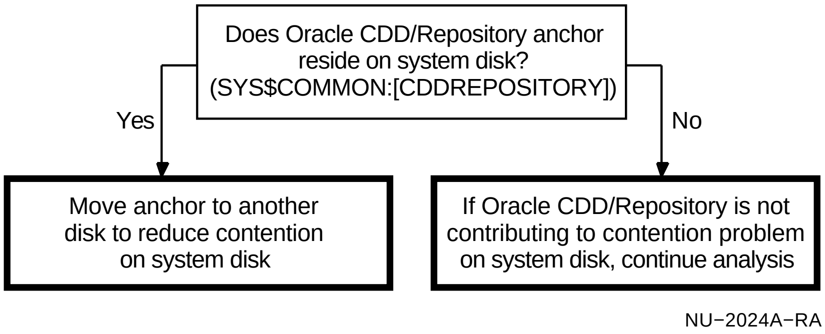
```
$ DEFINE RDMS$BIND_WORK_FILE WORK$DISK:[OFFLOADED_TEMP_FILES]
```

If the system disk remains a bottleneck after all the database files and repository files have been moved to other devices, it is possible that high page file activity is causing the bottleneck. If there is evidence of a high hard page fault rate and only a single page file is used, adding a secondary page file on another device may alleviate the bottleneck. This situation sometimes occurs in a database environment when process virtual memory requirements are much higher than their working set quotas. The database buffer parameters have the largest influence on process virtual memory needs.

8.1.2.1 Checking Oracle CDD/Repository

OpenVMS VAX OpenVMS Alpha If Oracle CDD/Repository is being used extensively, and the disk where the anchor directory is located is highly utilized, one option to reduce contention on that disk is to move the location of the CDD anchor to another disk. See Figure 8–3.

Figure 8–3 Decision Tree: Check Oracle CDD/Repository

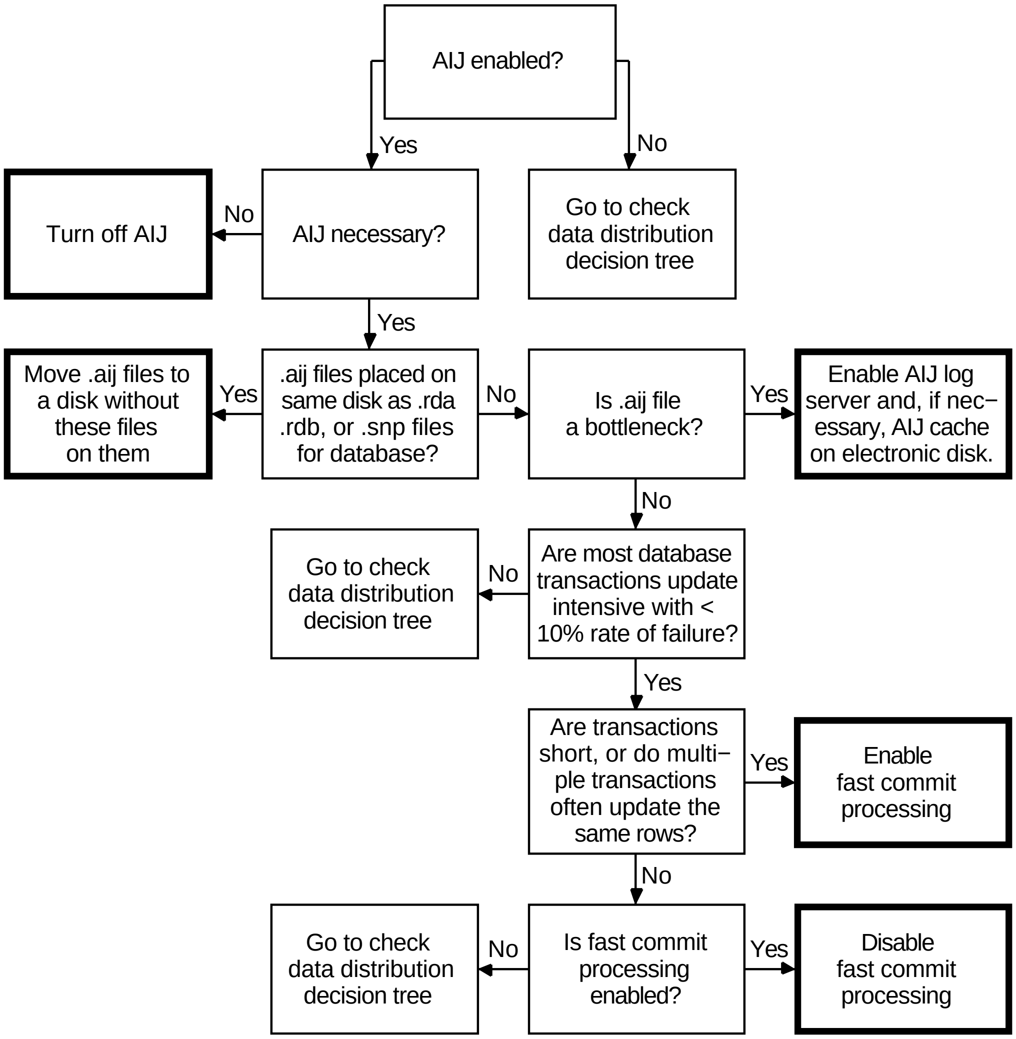

Does Oracle CDD/Repository anchor reside on system disk? (SYS$COMMON:[CDDREPOSITORY])
Yes
No
Move anchor to another disk to reduce contention on system disk
If Oracle CDD/Repository is not contributing to contention problem on system disk, continue analysis
NU−2024A−RA

For information on using Oracle CDD/Repository in a VMScluster environment, see Section 6.2.7, Oracle CDD/Repository Requirements. ♦

8.1.2.2 Checking AIJ

If after-image journaling is enabled, and the disk to which the .aij file is being written is highly utilized, one tuning possibility is to move the .aij file to a less utilized disk (see Figure 8–4). You cannot change the .aij file location using logical names or configuration parameters. Instead, use the SQL ALTER DATABASE statement to move the .aij file. Use the JOURNAL ALLOCATION IS and JOURNAL EXTENT IS clauses to further enhance performance of AIJ writes. These clauses ensure that the initial allocation of the journal file is large enough so that transactions do not need to wait for extents to occur.

Figure 8–4 Decision Tree: Check AIJ


AIJ enabled?
Yes
No
Turn off AIJ
No
AIJ necessary?
Go to check data distribution decision tree
Yes
Move .aij files to a disk without these files on them
Yes
.aij files placed on same disk as .rda .rdb, or .snp files for database?
No
Is .aij file a bottleneck?
Yes
Enable AIJ log server and, if nec−essary, AIJ cache on electronic disk.
No
Go to check data distribution decision tree
No
Are most database transactions update intensive with < 10% rate of failure?
Yes
Are transactions short, or do multi−ple transactions often update the same rows?
Yes
Enable fast commit processing
No
Go to check data distribution decision tree
No
Is fast commit processing enabled?
Yes
Disable fast commit processing

NU−2025A−RA

Another reason to move the .aij file to its own disk or to a nondatabase disk is that doing this makes your database more reliable. The .aij file allows the database administrator (DBA) to roll forward the database from the point of the last backup when a disk failure has occurred on one of the disks where an .rda, .rdb, or .snp file resided. For this reason, the .aij file should be on a separate disk from the .rda, .rdb, and .snp files for the database. If you use multiple .aij files for your database, each .aij file should be on a separate disk from the database's .rda, .rdb, and .snp files. Otherwise, if a disk failure occurs, you may not be able to recover.

It is usually a good idea to enable after-image journaling. Oracle Rdb recommends that AIJ journaling be enabled unless the database is read-only or it does not need to be recovered in case of failure.

You can improve database performance in some environments by enabling fast commit processing. When you enable fast commit, Oracle Rdb keeps updated pages in the buffer pool and does not write the pages to disk when a transaction commits. Updated pages remain in the buffer pool until they are written to the database as part of an asynchronous batch-write operation. Also, when a user-specified threshold (called a checkpoint) is reached, all the updated pages for multiple transactions are flushed to disk. If a transaction fails, all the previous transactions back to the last checkpoint must be redone because their updated pages have not been written to disk. Oracle Rdb uses information written to the .aij file to redo these transactions.

You should consider enabling fast commit transaction processing when you have a stable transaction processing environment and the following conditions exist: a process updates the same rows for multiple transactions; or, transactions are short and do not update many pages. Typically, these types of transactions tend to have few buffer pool overflows. The I/Os to the database are reduced for these types of transactions, which means the redo operation performed when a transaction fails and fast commit transaction processing is enabled will be shorter than the redo operation for long transactions that update many pages. See Section 4.1.5 for more information on fast commit transaction processing.

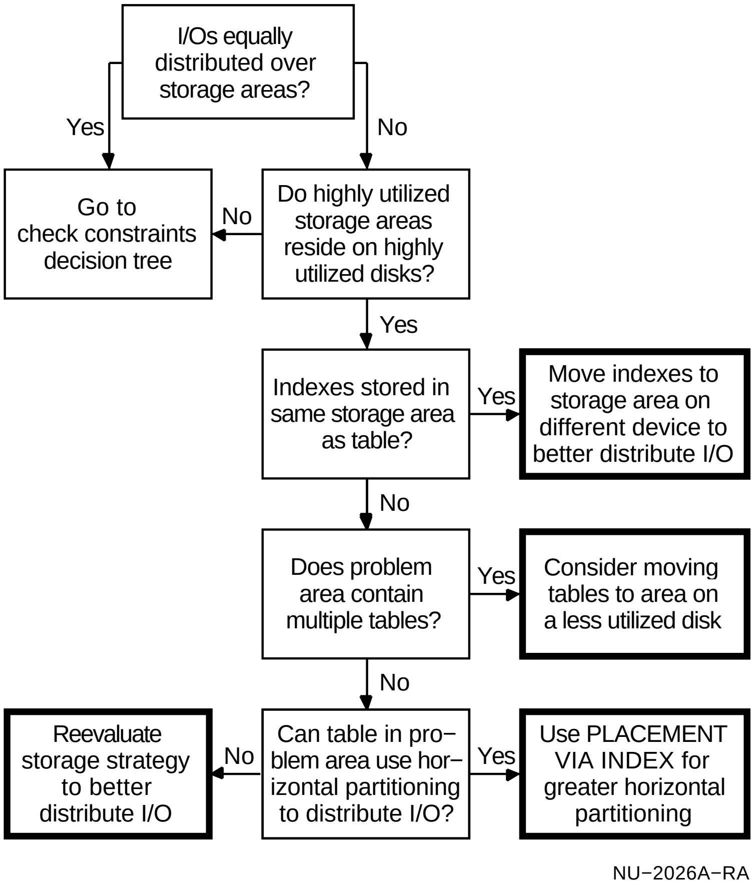
When you enable fast commit transaction processing, you can also enable journal optimization. This option can provide a significant increase in commit processing speed by eliminating the majority of I/O to the database root. This option enhances performance in database environments that are update-intensive.

For details on fast commit processing and AIJ optimization, refer to Section 4.1.5.

8.1.2.3 Checking Data Distribution

Figure 8–5 summarizes the steps you can take to check data distribution.

Figure 8–5 Decision Tree: Check Data Distribution


I/Os equally distributed over storage areas?
Yes
No
Go to check constraints decision tree
No
Do highly utilized storage areas reside on highly utilized disks?
Yes
Indexes stored in same storage area as table?
Yes
Move indexes to storage area on different device to better distribute I/O
No
Does problem area contain multiple tables?
Yes
Consider moving tables to area on a less utilized disk
No
Reevaluate storage strategy to better distribute I/O
No
Can table in pro– blem area use hor– izontal partitioning to distribute I/O?
Yes
Use PLACEMENT VIA INDEX for greater horizontal partitioning
NU−2026A−RA

One way to help determine I/O distribution across the database storage areas is to use the IO Statistics screen in the Performance Monitor. This gives you a general idea of how evenly distributed your I/O activity is across the storage areas and disks for a given time period. One way to measure this is to run the Performance Monitor for a period of time, enter the IO Statistics (by file) option, then bring up the Option menu and write a report. This generates a file (STATISTICS.RPT) that contains ASCII output of all the menus for that period.

You can edit this file using your favorite editor, or print it on any printer. In the STATISTICS.RPT file, you will find a section similar to the one shown in Example 8–3.

Example 8–3 IO Statistics (By File) Screen

```
Node: MYNODE        Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 14:24:45
Rate: 3.00 Seconds            File IO Statistics          Elapsed: 03:04:56.88
Page: 1 of 1     USER18:[PRODUCTION.DATABASE]PRODUCT_DB.RDB;1       Mode: Online
--------------------------------------------------------------------------------
                         For File: All data/snap files
statistic........... rate.per.second............. total....... average......
name................ max..... cur..... avg....... count....... per.trans....
total I/Os                       24                     39152          76.5
    (Synch. reads)        48       10        22.0       35254          68.9
    (Synch. writes)       16       14         2.4        3898           7.6
    (Extends)              0        0         0.0           0           0.0
    (Asynch. reads)        0        0         0.0           0           0.0
    (Asynch. writes)       0        0         0.0           0           0.0

statistic........... blocks.transferred.........  stall.time.(x100)...........
name................ avg.per.I/O.. total........  avg.per.I/O... total........
total I/Os                    5.3       209348              3.5       137914
    (Synch. reads)            5.7       199280              3.8       133681
    (Synch. writes)           2.6        10068              1.1         4233
    (Extends)                 0.0            0              0.0            0
    (Asynch. reads)           0.0            0              0.0            0
    (Asynch. writes)          0.0            0              0.0            0
--------------------------------------------------------------------------------
Exit Help Menu Options Reset Set_rate Unreset Write !
```

The total I/Os section, namely the maximum and average per second, and the total count values, can be used to determine the percentage of I/Os that are going to each area. If the Total I/O numbers are small, further optimization may yield only marginal results. Section 4.2.1.4 also provides information on the IO Statistics (by file) screen.

You also should look at the stall time values on the IO Statistics screen because long stall times cause the response time of an application to increase. When you are displaying statistics for all data files with the IO Statistics screen (as in Example 8–3), the value for the “stall time avg per I/O” field may not be what you expect if you have displayed the average stall time per I/O for the individual .rda and .snp files in the database. For example, it is possible for the value in the “stall time avg per I/O” field for the all data files screen to be lower than the average of the values for the “stall time avg per I/O” fields for the individual .rda and .snp files. Oracle Rdb issues write I/O operations in parallel, asynchronously (this is a batch-write mechanism). This means that the average stall time per I/O for the all data files screen is not the “average of the averages" shown for the individual .rda and .snp files; this would imply that the I/Os completed serially. Rather, the total I/O stall time is "the average of the averages divided by the number of I/Os"; the average for the all data files screen is usually a fraction of the individual file averages because the stall time is amortized across all I/Os issued in parallel.

For example, assume that Oracle Rdb does a batch-write operation to four storage areas (all in parallel, of course). Assume that each individual storage area's I/O operation takes 20 milliseconds. If the I/Os were done serially, then the average stall time for all storage areas would be 20 milliseconds. However, because the I/Os are done in parallel, the average for all areas is actually 5 milliseconds (20 milliseconds divided by 4 I/O operations).

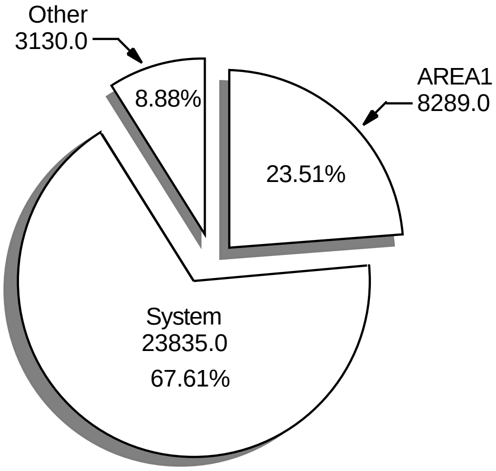
If asynchronous batch-write operations are enabled, Oracle Rdb performs write operations to the database, and does not have to wait for the write operation to complete. The I/Os for asynchronous writes are recorded on the IO Statistics (by file) screen, but no stall time is recorded on the IO Statistics (by file) screen for most of these writes because stalls are eliminated for most asynchronous batch-write operations. See Section 3.2.5 for more information on asynchronous batch-write operations.

If a disk takes 30 milliseconds for an I/O operation, then you should consider trying to improve (reduce) the stall times only when the average stall time per I/O on the IO Statistics (by file) screen is greater than 30 milliseconds. When the average stall time for an I/O operation is less than 30 milliseconds, this is attributable to the parallel writes performed by batch-write operations, as described earlier. You can examine disk performance statistics to monitor changes in disk performance.

Figure 8–6 shows the percentage of total read I/Os being generated by each storage area. This was calculated by dividing the total count read I/Os for each area by the total count read I/Os and multiplying the result by 100. The following table shows the total count read I/Os for all database files (see Example 8–3); the two storage areas (SYSTEM_DB and AREA1), which appear in Example 8–4 and Example 8–6, respectively; and other files, including .rdb, .snp, .ruj, and .aij files. The Other file I/O is calculated by subtracting all storage area total count read I/Os from the total.

	Read I/Os	% of Total Read I/Os
Total	35254	
System Area	23835	67.61
Area1	8289	23.51
Other	3130	8.88

Figure 8–6 Percentage Read I/Os by Storage Area



NU−2004A−RA

The total stall time column from all examples (Example 8–3 to Example 8–7) is useful for determining if read or write I/Os or both are a concern. The bigger the stall time, the slower the response time will be from the application. By enabling the asynchronous prefetch feature for a database, you can eliminate stall time for sequential read operations, as explained in Section 3.2.4.

The reports in Example 8–4 to Example 8–7 show data and snapshot file I/O measurements for those storage areas that are generating the greatest I/O. Because the system area is generating over 67% of the read I/Os, and the maximum values are high, the current storage structure could be causing a bottleneck. Further investigation into disk utilization with a view toward a better balance of I/Os across disks is necessary. Example 8–4 displays I/O statistics for the system storage area.

Example 8–4 I/O Statistics for the System Storage Area

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 14:29:37
Rate: 3.00 Seconds             File IO Statistics           Elapsed: 00:00:12.17
Page: 1 of 1      USER18:[PRODUCTION.DATABASE]PRODUCT_DB.RDB;1      Mode: Online
--------------------------------------------------------------------------------
            For File: USER07:[PRODUCTION.DATABASE.SYSTEM]SYSTEM_DB.RDA;1
statistic........... rate.per.second............ total....... average......
name................ max..... cur..... avg....... count....... per.trans....
total I/Os                          10                   25078          49.0
    (Synch. reads)         48        4      14.9        23835          46.6
    (Synch. writes)         6        6       0.8         1243           2.4
    (Extends)               0        0       0.0            0           0.0
    (Asynch. reads)         0        0       0.0            0           0.0
    (Asynch. writes)        0        0       0.0            0           0.0

statistic........... blocks.transferred.........  stall.time.(x100)...........
name................ avg.per.I/O.. total........  avg.per.I/O... total........
total I/Os                     5.5        138208             4.6        115420
    (Synch. reads)             5.7        135490             4.4        105137
    (Synch. writes)            2.2          2718             8.3         10283
    (Extends)                  0.0             0             0.0             0
    (Asynch. reads)            0.0             0             0.0             0
    (Asynch. writes)           0.0             0             0.0             0
--------------------------------------------------------------------------------
Exit Help Menu Options Reset Set_rate Unreset Write !
```

Example 8–5 displays I/O statistics for the system snapshot file.

Example 8–5 I/O Statistics for the System Snapshot File

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 14:35:21
Rate: 3.00 Seconds             File IO Statistics           Elapsed: 00:05:55.96
Page: 1 of 1     USER18:[PRODUCTION.DATABASE]PRODUCT_DB.RDB;1       Mode: Online
--------------------------------------------------------------------------------
            For File: USER19:[PRODUCTION.DATABASE.SYSTEM]SYSTEM_DB.SNP;1
statistic........... rate.per.second............ total....... average......
name................ max..... cur..... avg....... count....... per.trans....
total I/Os                           4                    1215           2.4
    (Synch. reads)          2        2        0.3          547           1.1
    (Synch. writes)         3        2        0.4          668           1.3
    (Extends)               0        0        0.0            0           0.0
    (Asynch. reads)         0        0        0.0            0           0.0
    (Asynch. writes)        0        0        0.0            0           0.0

statistic........... blocks.transferred.........  stall.time.(x100)...........
name................ avg.per.I/O.. total........  avg.per.I/O... total........
total I/Os                      4.9         5914              4.6         5631
    (Synch. reads)              6.0         3266              3.3         1809
    (Synch. writes)             4.0         2648              5.7         3822
    (Extends)                   0.0            0              0.0            0
    (Asynch. reads)             0.0            0              0.0            0
    (Asynch. writes)            0.0            0              0.0            0
--------------------------------------------------------------------------------
Exit Help Menu Options Reset Set_rate Unreset Write !
```

Example 8–6 displays I/O statistics for the AREA1 storage area.

Example 8–6 I/O Statistics for the AREA1 Storage Area

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 14:37:55
Rate: 3.00 Seconds             File IO Statistics          Elapsed: 00:08:29.96
Page: 1 of 1      USER18:[PRODUCTION.DATABASE]PRODUCT_DB.RDB;1      Mode: Online
--------------------------------------------------------------------------------
            For File: USER16:[PRODUCTION.DATABASE.AREA1]AREA1.RDA;1
statistic........... rate.per.second............. total....... average......
name................ max..... cur..... avg....... count....... per.trans....
total I/Os                           7                    9313          18.2
    (Synch. reads)         39        3        5.2         8289          16.2
    (Synch. writes)         5        4        0.6         1024           2.0
    (Extends)               0        0        0.0            0           0.0
    (Asynch. reads)         0        0        0.0            0           0.0
    (Asynch. writes)        0        0        0.0            0           0.0

statistic........... blocks.transferred.........  stall.time.(x100)...........
name................ avg.per.I/O.. total........  avg.per.I/O... total........
total I/Os                     5.3         49702             2.7         25104
    (Synch. reads)             5.7         47654             2.2         18545
    (Synch. writes)            2.0          2048             6.4          6559
    (Extends)                  0.0             0             0.0             0
    (Asynch. reads)            0.0             0             0.0             0
    (Asynch. writes)           0.0             0             0.0             0
--------------------------------------------------------------------------------
Exit Help Menu Options Reset Set_rate Unreset Write !
```

Example 8–7 displays I/O statistics for the AREA1 snapshot file.

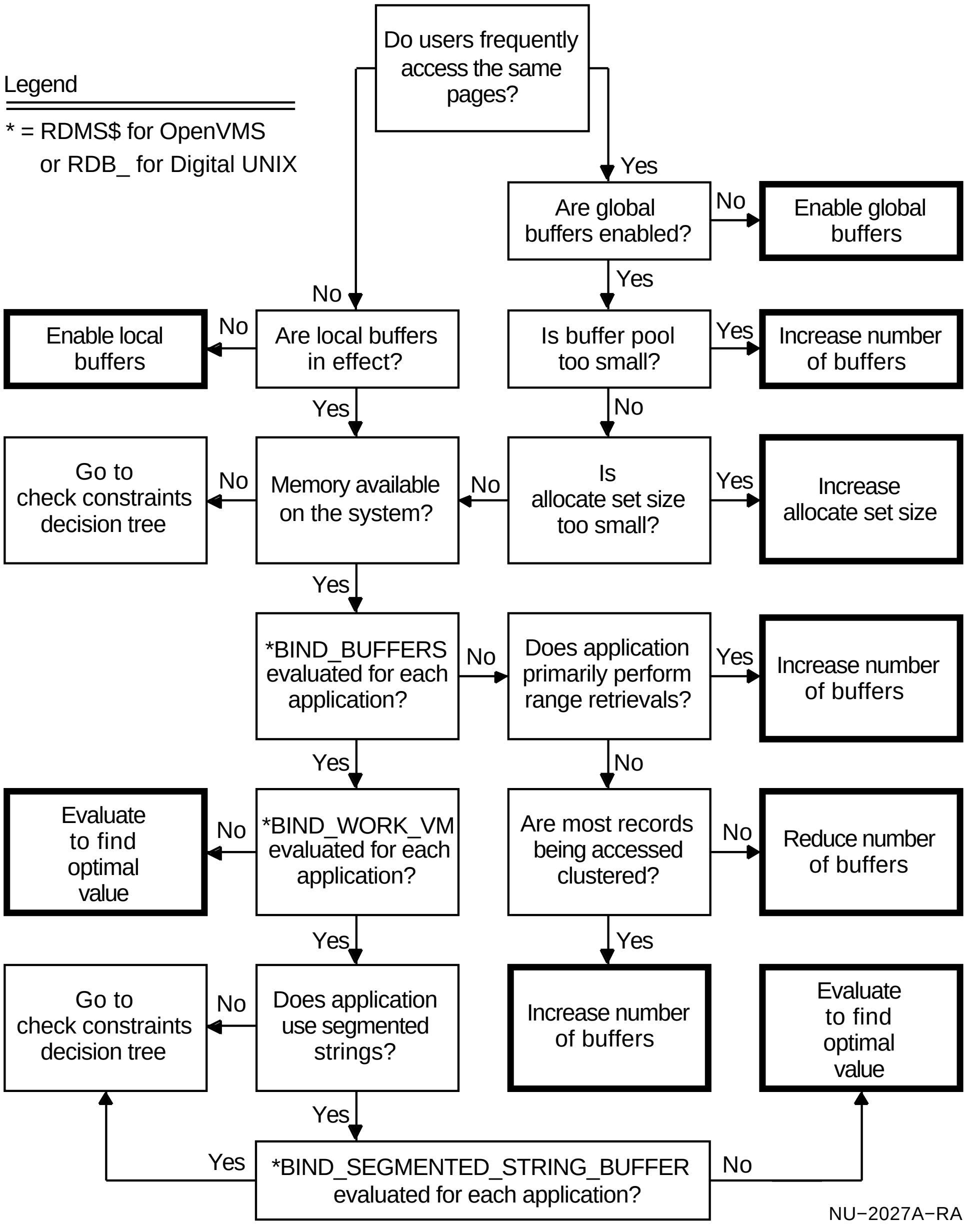
Example 8–7 I/O Statistics for the AREA1 Snapshot File

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 14:41:26
Rate: 3.00 Seconds             File IO Statistics          Elapsed: 00:12:00.67
Page: 1 of 1      USER18:[PRODUCTION.DATABASE]PRODUCT_DB.RDB;1      Mode: Online
-------------------------------------------------------------------------------
            For File: USER29:[PRODUCTION.DATABASE.AREA1]AREA1.SNP;1
statistic........... rate.per.second............. total....... average......
name................ max..... cur..... avg....... count....... per.trans....
total I/Os                           2                     797           1.6
    (Synch. reads)          1        1        0.2          254           0.5
    (Synch. writes)         3        1        0.3          543           1.1
    (Extends)               0        0        0.0            0           0.0
    (Asynch. reads)         0        0        0.0            0           0.0
    (Asynch. writes)        0        0        0.0            0           0.0

statistic........... blocks.transferred.........  stall.time.(x100)...........
name................ avg.per.I/O.. total........  avg.per.I/O... total........
total I/Os                     4.1          3306              5.9          4701
    (Synch. reads)             6.0          1524              4.0          1005
    (Synch. writes)            3.3          1782              6.8          3696
    (Extends)                  0.0             0              0.0             0
    (Asynch. reads)            0.0             0              0.0             0
    (Asynch. writes)           0.0             0              0.0             0
--------------------------------------------------------------------------------
Exit Help Menu Options Reset Set_rate Unreset Write !
```

8.1.3 Reducing I/O Operations

One way to reduce the number of direct I/O operations is to prevent the need for them. This can be done by making changes at the system, database, or application level. See Figure 8–7.

Figure 8–7 Decision Tree: Reduce I/O


Legend
* = RDMS$ for OpenVMS
or RDB_ for Digital UNIX
Do users frequently access the same pages?
Yes
No
Are global buffers enabled?
No
Enable global buffers
Yes
Are local buffers in effect?
No
Enable local buffers
Is buffer pool too small?
Yes
Increase number of buffers
Yes
No
Go to check constraints decision tree
No
Memory available on the system?
No
Is allocate set size too small?
Yes
Increase allocate set size
Yes
*BIND_BUFFERS evaluated for each application?
No
Does application primarily perform range retrievals?
Yes
Increase number of buffers
Yes
No
Evaluate to find optimal value
No
*BIND_WORK_VM evaluated for each application?
Are most records being accessed clustered?
No
Reduce number of buffers
Yes
Yes
Go to check constraints decision tree
No
Does application use segmented strings?
Increase number of buffers
Evaluate to find optimal value
Yes
Yes
*BIND_SEGMENTED_STRING_BUFFER evaluated for each application?
No
NU−2027A−RA

One way to control disk I/O is through buffer management. Oracle Rdb enables you to use local or global buffers. Refer to Section 4.1.2 for a general discussion of buffer management and the relative advantages of local and global buffers. The remainder of this section describes the use of local buffers and global buffers and how to tune them.

The most frequently used way to reduce I/O from the system level is to make the buffer pool size bigger, so that the disk I/Os are replaced by extra memory references. To do this, additional memory capacity must exist on the system. Because access to the buffer is much faster than access to the disk, this approach can cause substantial improvements in response time, especially when the access is on either slow or highly utilized disk devices.

One way to control the I/O buffer pool size is to change the number of buffers that the database maintains. This can be done in the database using the NUMBER OF BUFFERS clause of the SQL ALTER DATABASE statement, or from the system level through the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter. Because this parameter is dynamic, it can be modified with very little risk or cost. The default number of database buffers is 20.

OpenVMS VAX OpenVMS Alpha

The logical name can be set system-, group-, or process-wide, for finer buffer granularity. ♦

See Section 4.1.2.3 for additional information on the NUMBER OF BUFFERS clause.

To help quantify the additional memory requirement due to increased numbers of database buffers, a database application was run in 20-buffer increments and memory statistics collected. Figure 8–8 shows the results of this experiment. Because buffer size remained constant, increasing the number of buffers affected virtual memory consumption linearly, as expected. Changing the buffer size should move the line a constant factor upwards or downwards, depending on in which direction buffer size is changed. However, results can differ dramatically depending on the exact nature of the application being studied; therefore, you will need to run similar experiments on your database to determine the additional amount of memory consumed by increasing the number of buffers.

Figure 8–8 Virtual Memory Consumption Versus Number of Buffers

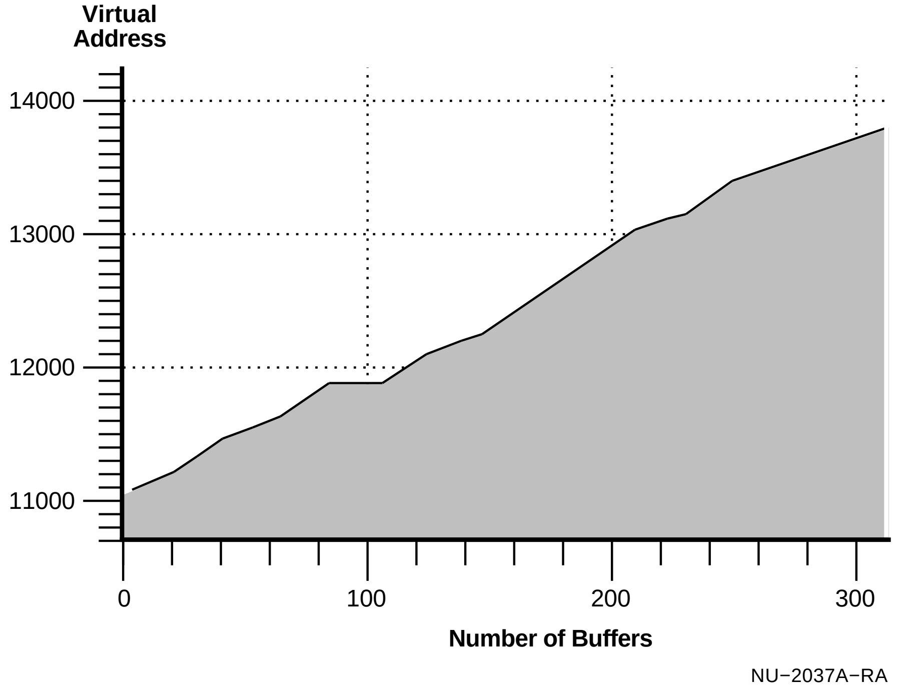
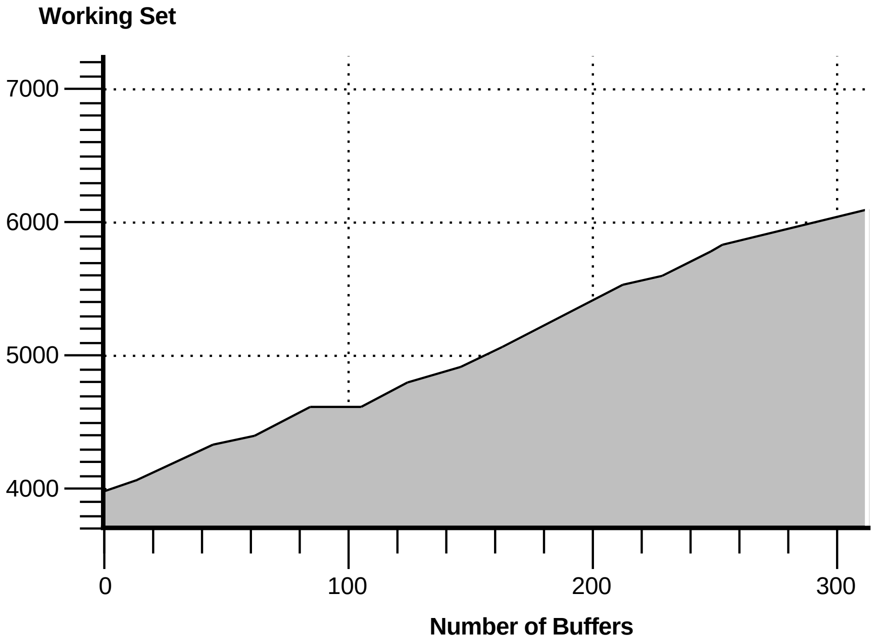


Figure 8–9 shows how working set size was affected by increasing the number of database buffers for the same application in 20-buffer increments. The results show working set consumption to be nearly linear as well.

Figure 8–9 Working Set Size Versus Number of Buffers



NU−2007A−RA

What does this mean for response time? Response time will vary depending on disk speed and utilization, the type of transaction the application is performing, the frequency with which updates occur, and the amount of contention in the database. As a general rule, applications that perform range retrievals benefit from having more buffers. This is also true if the data are clustered, as associated information will be read into the buffers when the first piece of information is accessed. Load-intensive workloads benefit from having small numbers of buffers. Because load-intensive workloads deal primarily with the physical database page, it makes little sense for them to incur the overhead of maintaining a large number of buffers.

Response time is not linear with respect to the number of database buffers.

One tangible way to measure buffer pool effectiveness database-wide is through the Performance Monitor PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches screens. The local buffer versions of these screens are shown in Example 8–8. The global buffer versions of these screens are shown in Example 8–9.

The way that you determine your database's buffer pool effectiveness depends on whether the database uses local buffers or global buffers.

When a database has local buffers enabled, the local buffer versions of the PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches screens are displayed by the Performance Monitor, as shown in Example 8–8. See the Performance Monitor help for more information about these screens.

Example 8–8 Local Buffer Versions of Performance Monitor PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches Screens

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 14:48:04
Rate: 3.00 Seconds        PIO Statistics--Data Fetches      Elapsed: 00:18:38.25
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1      Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

fetch for read              2287        0       20.4        18524         926.2
fetch for write               58        0        0.4          348          17.4

in LB: all ok               2035        0       18.3        16631         831.6
   LB: need lock             296        0        2.1         1949          97.5
   LB: old version             0        0        0.0            0           0.0

not found: read               34        0        0.3          292          14.6
         : synth               0        0        0.0            0           0.0

DAPF: success                  0        0        0.0            0           0.0
DAPF: failure                  0        0        0.0            0           0.0
DAPF: utilized                 0        0        0.0            0           0.0
DAPF: discarded                0        0        0.0            0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

(continued on next page)

Example 8–8 (Cont.) Local Buffer Versions of Performance Monitor PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches Screens

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 14:51:28
Rate: 3.00 Seconds        PIO Statistics--SPAM Fetches      Elapsed: 00:22:03.13
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

fetch for read                 4        0        0.1           59           3.0
fetch for write                0        0        0.0            0           0.0

in LB: all ok                  3        0        0.1           52           2.6
   LB: need lock               1        0        0.0            6           0.3
   LB: old version             0        0        0.0            0           0.0

not found: read                0        0        0.0            1           0.1
         : synth               0        0        0.0            0           0.0

DAPF: success                  0        0        0.0            0           0.0
DAPF: failure                  0        0        0.0            0           0.0
DAPF: utilized                 0        0        0.0            0           0.0
DAPF: discarded                0        0        0.0            0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For a database with local buffers enabled, you can calculate the percentage of page requests that find the page in the local buffer pool as follows:

```
( total number found in local buffer pool  / total number of page requests ) * 100
```

To calculate the “total number found in local buffer pool” count, add the sum of the “in LB: all ok” and “LB: need lock” counts on the PIO Statistics–Data Fetches screen to the sum of the “in LB: all ok” and “LB: need lock” counts on the PIO Statistics–SPAM Fetches screen. To calculate the “total number of page requests” count, add the sum of the “fetch for read” and “fetch for write” counts on the PIO Statistics–Data Fetches screen to the sum of the “fetch for read” and “fetch for write” counts on the PIO Statistics–SPAM Fetches screen.

In Example 8–8 the buffer pool effectiveness is as follows:

```
(18638 / 18931) * 100 = approximately 98.5%
```

In general, a higher percentage implies better buffering. If every page request found the page in the buffer pool, the buffer pool effectiveness would be 100%.

When a database has global buffers enabled, the global buffer versions of the PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches screens are displayed by the Performance Monitor, as shown in Example 8–9.

Example 8–9 Global Buffer Versions of Performance Monitor PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches Screens

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 15:47:52
Rate: 3.00 Seconds        PIO Statistics--Data Fetches      Elapsed: 00:00:46.03
Page: 1 of 1       RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....
fetch for read             1872        0       16.2        18524          926.2
fetch for write              39        0        0.3          348           17.4
in AS: all ok              1693        0       14.1        16101          805.1
   AS: lock for GB            0        0        0.0            0            0.0
   AS: need lock            127        0        1.3         1494           74.7
   AS: old version            0        0        0.0            0            0.0
in GB: need lock             73        0        1.0         1107           55.4
   GB: old version            0        0        0.0            0            0.0
   GB: transferred            0        0        0.0            0            0.0
not found: read              18        0        0.1          170            8.5
         : synth              0        0        0.0            0            0.0
DAPF: success                 0        0        0.0            0            0.0
DAPF: failure                 0        0        0.0            0            0.0
DAPF: utilized                0        0        0.0            0            0.0
DAPF: discarded               0        0        0.0            0            0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

(continued on next page)

Example 8–9 (Cont.) Global Buffer Versions of Performance Monitor PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches Screens

```
Node: MYNODE        Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 15:51:47
Rate: 3.00 Seconds       PIO Statistics--SPAM Fetches      Elapsed: 00:04:41.06
Page: 1 of 1      RDBVMS_USER1:[LOGAN.V70]MF_PERSONNEL.RDB;1       Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....
fetch for read                 5        0        0.1          59           3.0
fetch for write                0        0        0.0           0           0.0
in AS: all ok                  4        0        0.0          52           2.6
   AS: lock for GB             0        0        0.0           0           0.0
   AS: need lock               0        0        0.0           0           0.0
   AS: old version             0        0        0.0           0           0.0
in GB: need lock               1        0        0.0           6           0.3
   GB: old version             0        0        0.0           0           0.0
   GB: transferred             0        0        0.0           0           0.0
not found: read                0        0        0.0           1           0.1
         : synth               0        0        0.0           0           0.0
DAPF: success                  0        0        0.0           0           0.0
DAPF: failure                  0        0        0.0           0           0.0
DAPF: utilized                 0        0        0.0           0           0.0
DAPF: discarded                0        0        0.0           0           0.0

--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

For a database with global buffers enabled, you can calculate the following:

- The percentage of page requests that found the page in the allocate set
- The percentage of page requests that found the page in the global buffer pool
- The percentage of page requests that saved a disk I/O that would have occurred if your database had had local buffers enabled instead

You can calculate the percentage of page requests that find the page in the allocate set as follows:

```
( total number found in allocate set  / total number of page requests ) * 100
```

To calculate the “total number found in allocate set” count, add the sum of the “in AS: all ok,” “AS: lock for GB” and “AS: need lock” counts on the PIO Statistics–Data Fetches screen to the sum of the “in AS: all ok,” “AS: lock for GB” and “AS: need lock” counts on the PIO Statistics–SPAM Fetches screen. To calculate the “total number of page requests” count, add the sum of the “fetch for read” and “fetch for write” counts on the PIO Statistics–Data Fetches screen to the sum of the “fetch for read” and “fetch for write” counts on the PIO Statistics–SPAM Fetches screen.

In Example 8–9, the percentage of page requests that find the page in the allocate set is as follows:

```
(17647 / 18931) * 100 = approximately 93.2%
```

You can calculate the percentage of page requests that find the page in the global buffer pool (this includes the pages found in the allocate set) as follows:

```
( total number found in global buffer pool / total number of page requests ) * 100
```

To calculate the “total number found in global buffer pool” count, add the sum of the “in AS: all ok,” “AS: lock for GB,” “AS: need lock,” and “In GB: need lock” counts on the PIO Statistics–Data Fetches screen to the sum of the “in AS: all ok,” “AS: lock for GB,” “AS: need lock,” and “In GB: need lock” counts on the PIO Statistics–SPAM Fetches screen. To calculate the “total number of page requests” count, add the sum of the “fetch for read” and “fetch for write” counts on the PIO Statistics–Data Fetches screen to the sum of the “fetch for read” and “fetch for write” counts on the PIO Statistics–SPAM Fetches screen.

In Example 8–9, the percentage of page requests that found the page in the global buffer pool (including the allocate set) is as follows:

```
(18760 / 18931) * 100 = approximately 99.1%
```

After calculating the allocate set effectiveness and buffer pool effectiveness, you can determine whether the number of global buffers allocated to each process (allocate sets) and the number of buffers in the buffer pool are appropriate. If you want to increase the buffer pool effectiveness for the database, and you have additional memory capacity on the system, increase the number of buffers in your buffer pool. This should increase the buffer pool effectiveness.

If the buffer pool effectiveness is satisfactory but the allocate set effectiveness is low, you can increase the number of buffers allocated to processes, which should increase the allocate set effectiveness. Performance is better if a page is found in a user’s allocate set instead of in the global buffer pool. Be careful not to increase the size of allocate sets too much. For example, suppose a database has a global buffer pool of 1000 buffers, the default allocate set size for each process is 30 buffers, and 20 user processes must be able to access the database. With the 1000 buffers in the buffer pool and 20 processes that need to access the database, you should not make the default allocate set size greater than 50 buffers; when an allocate set size of 50 buffers is defined, each of the 20 user processes that needs to access the database can use all 50 buffers allocated to it. See Section 4.1.2.2 for more information on how Oracle Rdb determines the number of buffers to allocate to each process.

If your database has global buffers enabled, you might wonder what percentage of your page requests saved a disk I/O that would have occurred if your database had local buffers enabled instead. To calculate the percentage of page requests that would have resulted in I/O operations if your database had had local buffers enabled instead of global buffers enabled, use the following formula. For each field in the formula, provide the sum of the counts for the field from the PIO Statistics–Data Fetches and PIO Statistics–SPAM Fetches screens:

```
( 1 - ( AS: old version + in GB: old version + not found: read + : synth ) /
( AS: old version + in GB: need lock + in GB: old version +
  not found: read + : synth ) ) * 100
```

In Example 8–9, the percentage of page requests that saved a disk I/O that would have occurred if the database had had local buffers enabled is as follows:

```
(1 -  171 / 1284) * 100 = approximately 86.7%
```

To determine the number of global buffers that you will need to achieve a certain buffer pool effectiveness for a database, increase the size of the buffer pool by different amounts and calculate the percentage of page requests that find the page in the buffer pool for each increasingly larger global buffer pool.

OpenVMS VAX OpenVMS Alpha

When you set the RDM$BIND_BUFFERS logical name (also discussed in Section A.23), you need to address several system issues. First, you may need to update the UAF quotas for the account from which the database application will be run. DIOLM must be greater than or equal to the number of database buffers. ASTLM should be at least 12 greater than DIOLM. ENQLM may need to be increased, because more locks will be used for the additional buffers. Memory parameters BYTLM and PGFLQUOTA may need to be increased to handle the increased memory requirements. Additionally, working set parameters WSDEFAULT, WSQUOTA, and WSEXTENT may need to be adjusted to allow the processes to have more memory. If increasing the number of buffers causes a large increase in the page fault rate, adjusting the working set limit for the process should resolve the problem. Finally, the SYSGEN parameter VIRTUALPAGECNT may need to be increased before the process can take advantage of the additional buffers. If increasing the number of buffers does not result in an increased consumption of virtual memory, chances are good that you have reached the VIRTUALPAGECNT limit. ♦

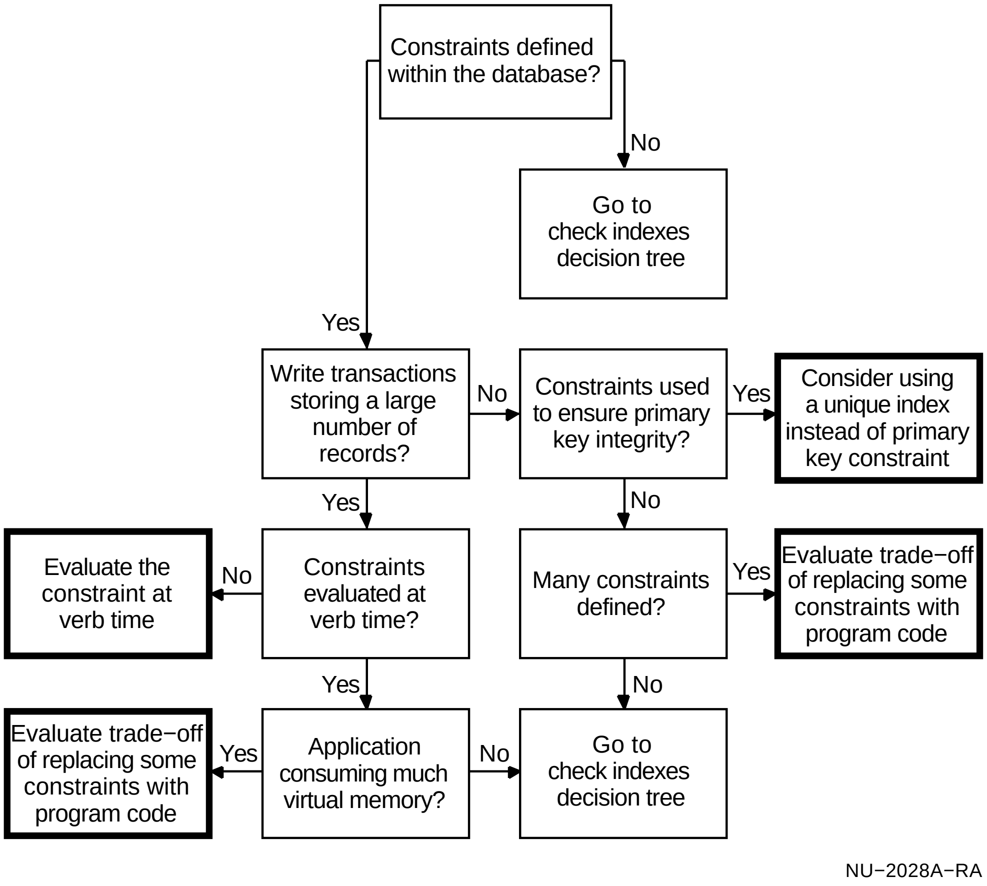
If RDM$BIND_BUFFERS has already been optimized, you can use the RDMS$BIND_WORK_VM logical name or the RDB_BIND_WORK_VM configuration parameter to reduce I/O activity (also discussed in Section A.92). RDMS$BIND_WORK_VM and RDB_BIND_WORK_VM can be used to reduce disk I/O operations for database matching operations by allowing you to specify the amount of virtual memory (VM) in bytes that will be allocated to your process for use in matching operations. The default is 10,000 bytes. Normally, when the 10,000 bytes of VM is exhausted, Oracle Rdb uses a temporary file for any additional data. Specifying larger values can reduce I/O by eliminating or reducing the need for these temporary files to be created.

OpenVMS VAX OpenVMS Alpha The value of RDMS$BIND_WORK_VM should not be greater than the UAF parameter PGFLQUOTA. ♦

8.1.3.1 Checking Constraints

If constraints are defined within the database, understanding the transactions can help you understand how the constraints may be using resources (see Figure 8–10). For example, if a long transaction stores a large number of rows, virtual memory resources may be consumed. In this situation, if the constraint can be evaluated at verb time rather than at commit time, fewer memory resources will be consumed. The default is to evaluate the constraint at commit time.

Figure 8–10 Decision Tree: Check Constraints

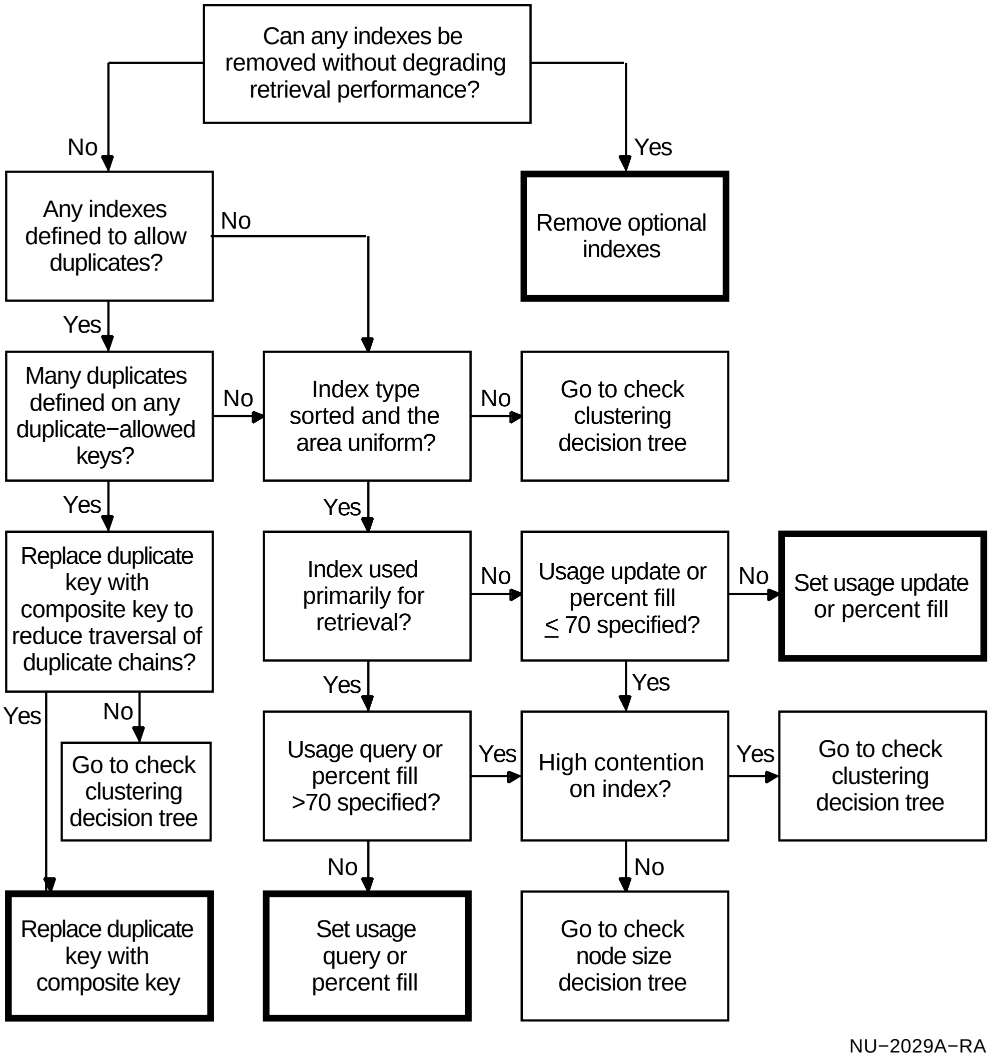

Constraints defined within the database?
No
Go to check indexes decision tree
Yes
Write transactions storing a large number of records?
No
Constraints used to ensure primary key integrity?
Yes
Consider using a unique index instead of primary key constraint
Yes
No
Evaluate the constraint at verb time
No
Constraints evaluated at verb time?
Many constraints defined?
Yes
Evaluate trade−off of replacing some constraints with program code
Yes
No
Evaluate trade−off of replacing some constraints with program code
Yes
Application consuming much virtual memory?
No
Go to check indexes decision tree
NU−2028A−RA

If you suspect that constraints are degrading the overall performance of your database environment, the easiest way to verify this is to remove the constraints and see how much performance improves. It may be that more constraints exist than can be optimally handled by the current resources. In other cases, replacing the constraint with an index that performs the same function may improve performance (for example, replacing a primary key constraint with a unique index).

8.1.3.2 Checking Indexes

Figure 8–11 summarizes the steps you can follow to see how indexes are affecting performance.

Figure 8–11 Decision Tree: Check Indexes



Can any indexes be removed without degrading retrieval performance?
No
Yes
Any indexes defined to allow duplicates?
No
Remove optional indexes
Yes
Many duplicates defined on any duplicate−allowed keys?
No
Index type sorted and the area uniform?
No
Go to check clustering decision tree
Yes
Yes
Replace duplicate key with composite key to reduce traversal of duplicate chains?
Index used primarily for retrieval?
No
Usage update or percent fill ≤ 70 specified?
No
Set usage update or percent fill
Yes
No
Yes
Yes
Go to check clustering decision tree
Usage query or percent fill >70 specified?
Yes
High contention on index?
Yes
Go to check clustering decision tree
No
No
Replace duplicate key with composite key
Set usage query or percent fill
Go to check node size decision tree
NU−2029A−RA

Refer also to the section on setting sorted index characteristics in the *Oracle Rdb7 Guide to Database Design and Definition* for additional information.

If a number of indexes are defined for a table, the overhead of maintaining those indexes can degrade update performance. The trade-off here is query performance versus load performance. You must decide if it is more important to get data into the database quickly, or to be able to generate a greater number of queries quickly.

If load performance is the priority, and indexes are defined that are not used by the standard reports, consider removing them to reduce overhead during store operations. If an index has many duplicates, removing it reduces overhead even more. Note that if the indexes not used by the standard reports are removed, the DBA should take care that users do not perform ad hoc queries that would normally use the removed indexes; without the availability of the usual indexes, these queries might be forced to use sequential access to tables, which could cause contention problems.

An alternative to removing indexes is to combine them. This approach can reduce the number of I/Os required to maintain the indexes. For example, assume you have three indexes: LAST_NAME_INDEX, FIRST_NAME_INDEX, and AGE_INDEX. If you combine these three indexes into one index, LAST_NAME_FIRST_NAME_AGE_INDEX, and delete the original three indexes, overhead will be reduced. The storage space for one index definition and one index structure is less than for three indexes, and I/O and locking need to be performed on only one index instead of three. Note that you should combine indexes like this only if all database queries specify the first index segment of the resulting combined index, instead of any of the other segments:

```
SQL> SELECT * FROM CONTRACTORS WHERE LAST_NAME = 'Jones';
```

Any database queries that request retrievals from any index segments besides the first in the combined index may perform poorly.

Another design consideration when you examine index overhead during a tuning analysis session is how the index is being used. If the index is defined on a column used primarily for direct match retrieval, and you have a sorted index defined to handle the query, creating a mixed area with a hashed index might be more efficient. For example, an index like PART_NUMBER, where you perform an exact match against the part number to fetch the row, might qualify for this type of change.

A final consideration is sorted index definition. If you know that an index is used primarily for efficient loading of data, you can specify usage update in the index definition. Usage query can be specified if the index is to be used primarily for queries. These options set the fullness percentage on each index node. Update sets the fullness percentage to 70, query sets the fullness percentage to 100. Alternately, percent fill can be specified during the definition if a finer granularity of control is required. If both usage and percent fill are specified, the usage parameter takes precedence.

The Performance Monitor has several screens (described in Section 3.9.5.2) that show information about indexes. These can help you determine how extensively duplicate indexes are being used, how heavily indexes are being used for insertion, removal, and retrieval, and other similar issues.

It is important to note that unique indexes are more efficient than duplicate indexes because cardinalities for unique indexes do not need to be maintained in the system tables. This reduces I/O and locking for unique indexes.

The RMU Analyze Indexes command shown in Example 8–10 can provide you with physical information about the index structure, such as the number of levels (max level) and number of records. The more levels an index has, the more index nodes Oracle Rdb must access to find the data record. Therefore, higher levels usually mean more I/O activity. Refer to Section 3.9.5.1 for additional information.

Example 8–10 Using the RMU Analyze Indexes Command

```
$ RMU/ANALYZE/INDEX PRODUCTION$DB_LOGICAL

---------------------------------------------------------------------------------------

 Indices for database  - USER18:[PRODUCTION.DATABASE]PRODUCT_DB.RDB;

---------------------------------------------------------------------------------------
 Index PTHIST$DATE_TIME for relation PART_HISTORY duplicates allowed
Max Level: 4, Nodes: 1051, Used/Avail: 246237/418298 (59%), Keys: 20770, Records: 10349
  Duplicate nodes: 9371, Used/Avail: 149936/244528 (61%), Keys: 9371, Records: 18742

---------------------------------------------------------------------------------------
 Index PTHIST$IDET_DATE for relation PART_HISTORY duplicates allowed
 Max Level: 2, Nodes: 18, Used/Avail: 3857/7164 (54%), Keys: 471, Records: 12
     Duplicate nodes: 910, Used/Avail: 232632/304534 (76%), Keys: 442, Records: 29079

---------------------------------------------------------------------------------------
   .
   .
   .
```

The RMU Analyze Placement command shown in Example 8–11 can provide you with information about how indexes have placed data rows in a storage area and how accessible the data rows are. This information includes the maximum and average number of index records accessed to reach a data row, the total number of pages traversed to reach a data row, and considering the buffer size, whether or not the index and data rows would both be in the buffer. Refer to Section 4.3.1 for more information on the RMU Analyze Placement command.

Example 8–11 Using the RMU Analyze Placement Option=Normal Command

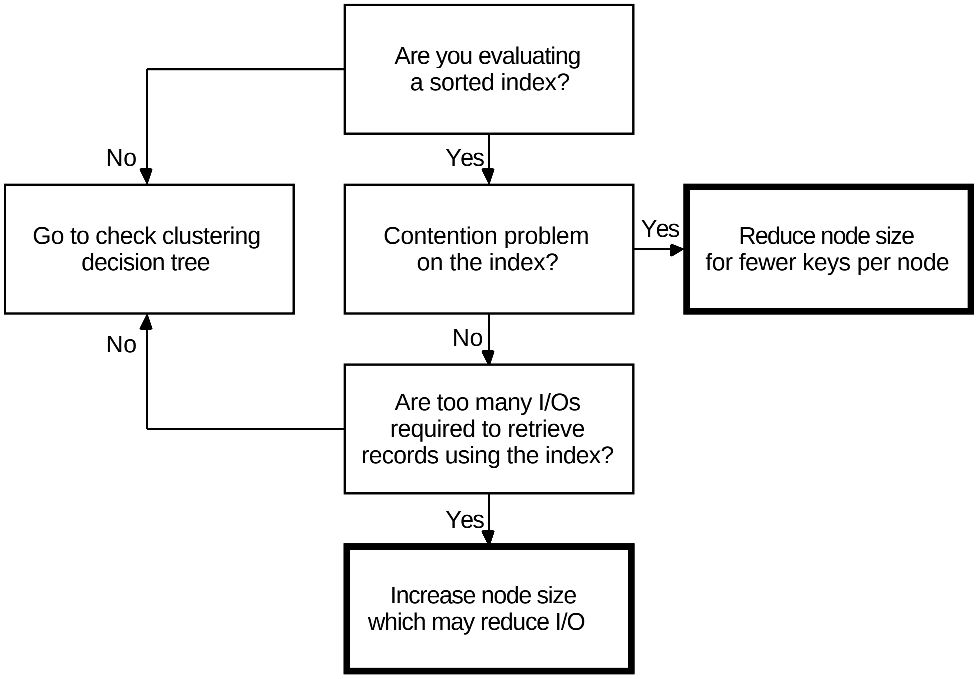
```
$ RMU/ANALYZE/PLACEMENT PRODUCTION$DB_LOGICAL PTHIST$DATE_TIME /OPTION=NORMAL
------------------------------------------------------------------------------
 Indices for database  - USER18:[PRODUCTION.DATABASE]PRODUCT_DB.RDB;

------------------------------------------------------------------------------
 Sorted Index PTHIST$DATE_TIME for relation PART_HISTORY duplicates not allowed
 Levels: 4, Nodes: 1051, Keys: 20770, Records: 10349
 Maximum path length --  DBkeys: 5, IO range: 1 to 5
 Average path length --  DBkeys: 4.20, IO range: 2.10 to 3.90

------------------------------------------------------------------------------
```

This information is useful in determining whether there are any performance problems related to rows stored using an index for which the PLACEMENT VIA INDEX clause is specified in the table's storage map. Any problems found might indicate that the page size is too small, the storage area is too small, or you are running out of space to store data rows and index records.

8.1.3.3 Checking Node Size

Figure 8–12 summarizes the steps you can take to check the node size.

Figure 8–12 Decision Tree: Check Node Size



Are you evaluating a sorted index?
No
Yes
Go to check clustering decision tree
Contention problem on the index?
Yes
Reduce node size for fewer keys per node
No
No
Are too many I/Os required to retrieve records using the index?
Yes
Increase node size which may reduce I/O

NU−3046A−RA

When checking whether index nodes are sized correctly, you should first determine whether a contention problem due to updates exists on the index nodes. If a contention problem does exist, decrease the node size for the index. When you reduce the node size, fewer index keys fit in each node, and fewer keys are locked by each update operation, which can alleviate the contention problem. It may not be possible to determine that contention is caused by a particular index; therefore, you can instead try to confirm that contention is not a problem. To do this, select the "stall time x 100" screen of the Performance Monitor Locking (one stat field) screen and look at the statistics for record locks. If this value is low, it indicates that contention in the index is not a problem, and the index nodes are not too large.

If the problem is that too many I/Os are required to fetch a database record, you should determine how many index node levels exist and, if the index is in its own storage area, how many I/Os per transaction are occurring in the area. You can sometimes reduce the number of I/Os by increasing the size of index nodes. When the size of index nodes is increased, the number of index node levels is sometimes reduced, which can reduce the number of I/Os needed to fetch a database record using the index. Use the RMU Analyze Placement command (see Example 8–11) to display the number of index node levels in an index. The number of I/Os required to fetch a record using a sorted index is:

```
Number of I/Os = Number of Node Levels + Number of Duplicate Nodes on
                 Other Pages - Number of Index Node Levels in the
   Buffer Pool + an I/O to Retrieve the Requested Record
```

Using the formula for a sorted index with 5 node levels, 2 of which fit in the buffer pool when the index is used, and no duplicate nodes, the number of I/Os required to fetch a record with the index is 4:

```
     4    =    5   +   0   -   2  +  1
```

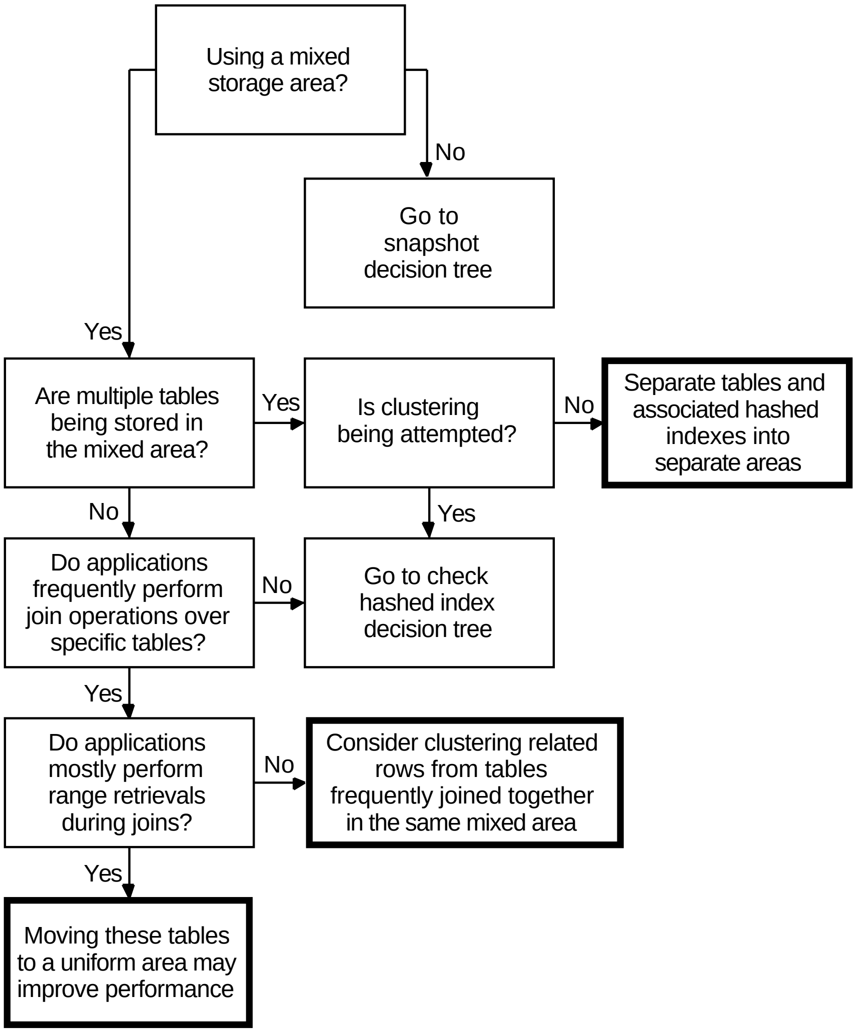
If the index is in its own storage area, use the IO Statistics (by file) screen to find the average number of I/Os per transaction in the storage area. If this value is high, increasing the size of the nodes can reduce the number of node levels and the amount of I/Os required to fetch records.

If you have infrequent updates to a table, and an index is used primarily for queries, specify a high value with the PERCENT FILL clause of the SQL ALTER INDEX or CREATE INDEX statements. Specifying a high percent fill value will result in an index with fewer levels, which should reduce the number of intermediate nodes fetched and the number of I/Os.

8.1.3.4 Checking Clustering

If you are using mixed storage areas, under certain conditions you can use record clustering to reduce I/O (see Figure 8–13). For example, if your application frequently performs join operations over several tables for exact match queries, clustering the records together in the same storage area is an option for I/O savings. In other words, inter-table clustering is good for data that will be retrieved by exact match join operations because the related data are clustered together, so fewer I/Os need to be performed.

Figure 8–13 Decision Tree: Check Clustering



Using a mixed storage area?
No
Go to snapshot decision tree
Yes
Are multiple tables being stored in the mixed area?
Yes
Is clustering being attempted?
No
Separate tables and associated hashed indexes into separate areas
No
Yes
Do applications frequently perform join operations over specific tables?
No
Go to check hashed index decision tree
Yes
Do applications mostly perform range retrievals during joins?
No
Consider clustering related rows from tables frequently joined together in the same mixed area
Yes
Moving these tables to a uniform area may improve performance

NU−2031A−RA

If the PLACEMENT VIA INDEX clause has been specified for a sorted index in a table's storage map, you can use that sorted index to store records in a particular order. This can lead to performance improvements if there are queries that require ordered retrieval of data. For example, if your applications always report information by ascending lot number, you can sort the records in ascending order, then store them using a sorted index on a LOT_ID column with an ascending key. This causes Oracle Rdb to attempt to store the records in ascending alphanumeric order. The I/O savings from this technique are due to the “clustering” of records near each other. For example, assume 20 parts comprise a lot, 10 parts fit on a database page, and queries often appear in the following form:

```
SQL> SELECT LOT_ID, PART_ID FROM LOTS
cont>   WHERE LOT_ID > 891001;

LOT_ID            PART_ID

891001-ABCDE      00034443
891001-ABCDE      00034444
   .
   .
   .
891001-CWEEW      00355344
```

If 1000 lots are in the database and 10 of them satisfy the query, and if the records are clustered as just described, then Oracle Rdb can access the records in approximately 20 I/O operations (2 I/Os per lot multiplied by 10 lots). If the data is not clustered, the worst case is 2000 I/O operations for the same request (assuming the records are so scattered that every part_id from the lot is on a different page, far enough away from the target page to not get pulled into a buffer). Note that clustering should be used only if you can accurately predict the maximum number of records that will be stored on each page.

Clustering with hashed indexes is also done using indexes for which the PLACEMENT VIA INDEX clause has been specified in the table’s storage map. As stated earlier, if join operations occur frequently between two or more tables for direct match queries, clustering the associated records in the same storage area will retrieve the records with I/O.

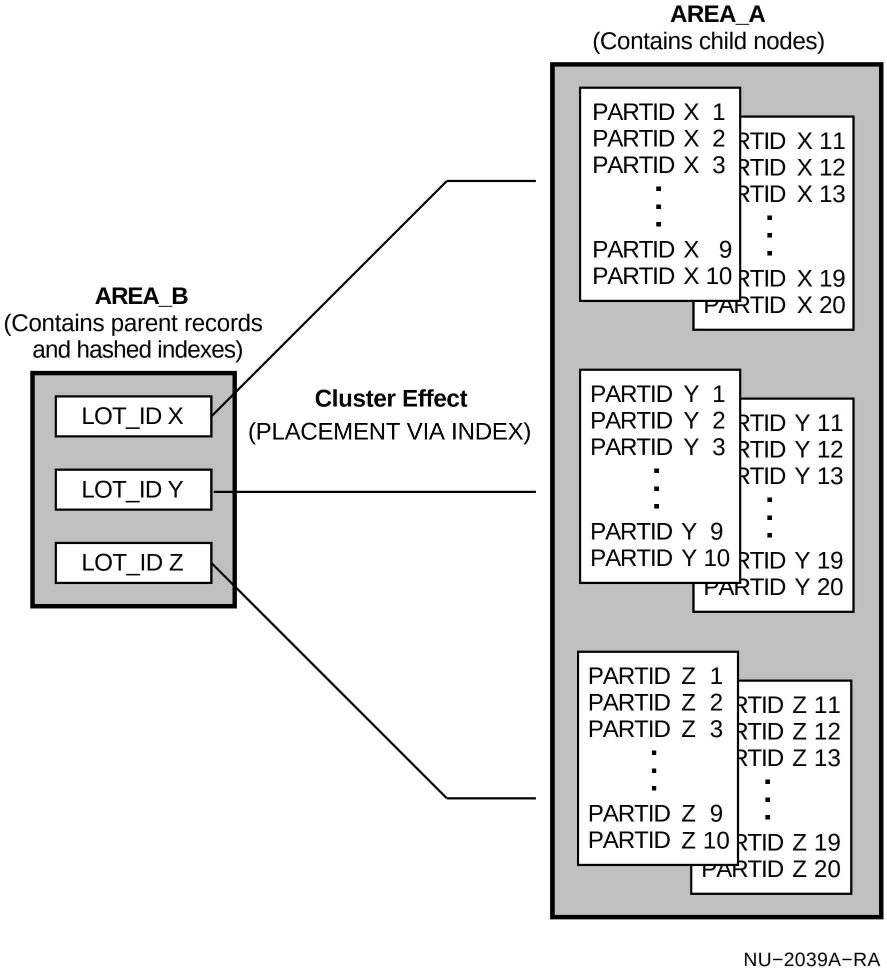
Similarly, when records are retrieved based on a parent-child relationship (one to many), clustering with hashed indexes could improve performance. In manufacturing, this is a common relationship between a lot ID or batch ID and the parts that are in it. If this relationship exists in your application and you would like to cluster the records together in the same storage area, carefully compute the values of certain storage area parameters such as allocation, page size, and SPAM threshold to avoid performance degradation due to file extending, page overflow, and fragmentation, respectively.

If it is impractical or physically impossible to group the records in one storage area, you can use shadow pages to gain a cluster effect across two storage areas. With this approach, your parent records (lot records) and both hashed indexes are stored in one storage area, and the child nodes (part records) are stored together in a second area. For example:

```
create storage map LOTID_MAP             create unique index LOTID_HASH
   for WIP_LOTS                             on WIP_LOTS ( LOT_ID )
   store in AREA_B                          store in AREA_B
   placement via index LOTID_HASH;          type is HASHED;

create storage map PARTID_MAP            create index PARTID_HASH
   for PART_INFO                            on PART_INFO ( LOT_ID )
   store in AREA_A                          store in AREA_B
   placement via index PARTID_HASH;         type is HASHED;
```

This approach works by storing the PART_INFO shadow records in storage area A at the same relative offset as those in storage area B, but not necessarily on the same page numbers (see Figure 8–14). For another example, refer to Section 3.9.7.3.

Figure 8–14 Using Shadow Pages for Clustering

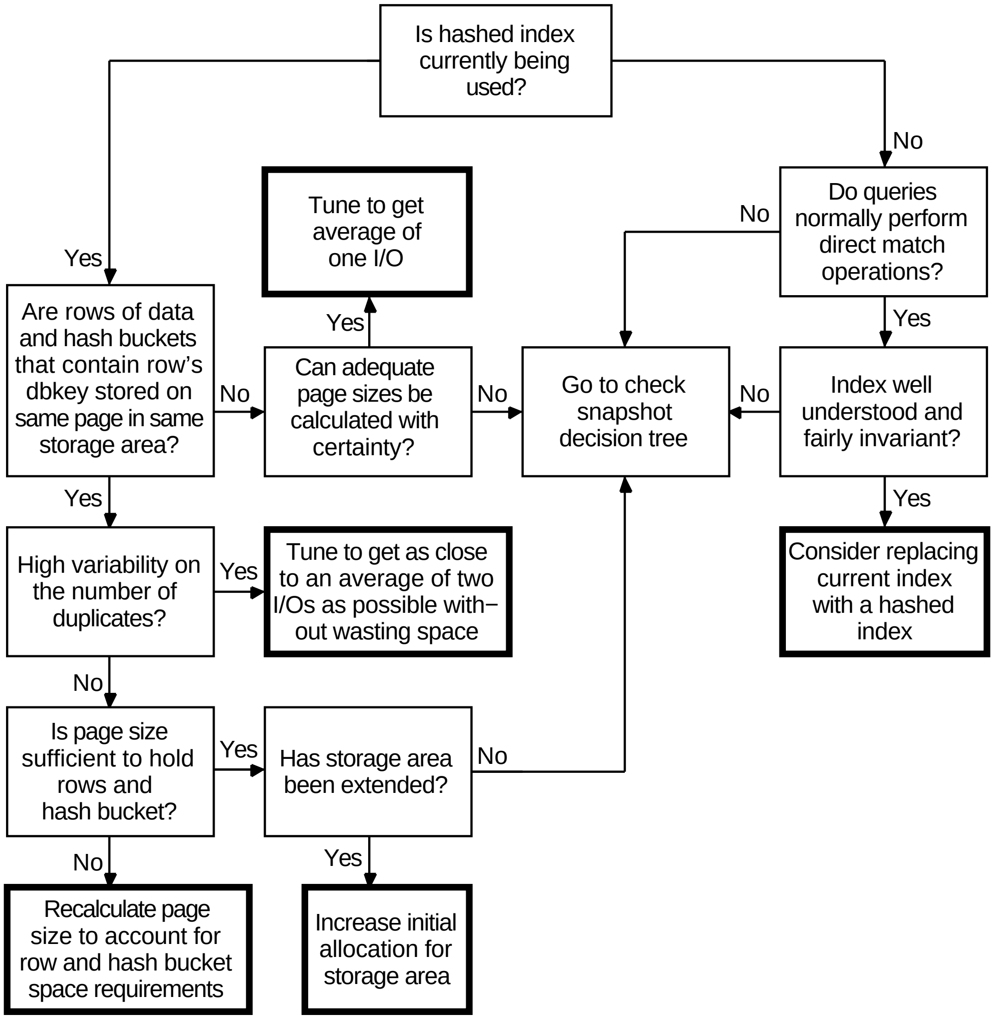


Oracle Corporation does not recommend storing miscellaneous tables and sorted indexes in a mixed storage area with hashed records or hash buckets. If clustering is not the goal, isolating hashed indexes and records removes the risk that some other database entity will use up the space on the page and ruin your carefully calculated page sizes.

8.1.3.5 Checking Hashed Indexes

Figure 8–15 summarizes the steps you can take to check the effect of hashed indexes on performance.

Figure 8–15 Decision Tree: Check Hashed Index


Is hashed index currently being used?
Yes
No
Do queries normally perform direct match operations?
No
Yes
Tune to get average of one I/O
Are rows of data and hash buckets that contain row's dbkey stored on same page in same storage area?
No
Can adequate page sizes be calculated with certainty?
Yes
No
Go to check snapshot decision tree
No
Index well understood and fairly invariant?
Yes
Yes
High variability on the number of duplicates?
Yes
Tune to get as close to an average of two I/Os as possible with– out wasting space
Consider replacing current index with a hashed index
No
Is page size sufficient to hold rows and hash bucket?
Yes
Has storage area been extended?
No
No
Yes
Recalculate page size to account for row and hash bucket space requirements
Increase initial allocation for storage area
NU–2032A–RA

To determine if hashed or sorted indexes are best for a particular table requires an understanding of the queries that will be performed on that data. If the requests generally are direct match queries, and the index can be characterized fairly well, defining a hashed index on the table can access the data with optimal results.

When hashed indexes are defined, a number of parameters are critical to how well the index performs. These parameters include: the initial allocation of the storage area, the page size, the record size, a storage strategy, SPAM information, key size, an estimate of the number of unique keys and the number of duplicates, and an estimate of the total number of records associated with the hashed index.

- The initial allocation of the storage area

  Extents in a mixed storage area seriously impair performance when a hashed index is being used. Because the hash key is not recalculated to account for the new space, the target page where the insertion is to be attempted is likely to be full. Additional I/O is incurred to find a free page to insert the record.

  If you have an undersized storage area and are mass loading data using a hashed index, the load time is greatly increased because, as the storage area fills, Oracle Rdb spends more and more time looking for free space on data pages that are further and further from the target page. If you have several undersized storage areas being loaded in succession, the problem becomes even more pronounced.

- The page size

  It is critical that you do not underestimate the page size when you use hashed indexes. Perform the calculations based on the uncompressed size of user-stored data records.

  Table 8–1 provides information to help you size a data page. Refer to the *Oracle Rdb7 Guide to Database Design and Definition* for more detailed information on this topic.

Table 8–1 Estimating the Number of Bytes per Entry Plus Overhead Bytes for Each Respective Index Record Type on a Data Page

Category	Bytes per Entry	Total
SYSTEM RECORD		
No. of hashed indexes[1]	a	a
Total system record size		
Overhead	4	4
Minimum	6	(6*a)+4
Maximum[2]	10	(10*a)+4
HASH BUCKET		
Total hash bucket entry size		
Key size[3]	1	1
Key length[4]	k+1	k+1
Overhead/entry[5]	12	12
Total/entry[6]	12+1+k+1=b	b
No. of entries[7]	c	c
Overhead/bucket[8]	13	13
Total bucket size	(b*c)+13	(b*c)+13
DUPLICATE NODE RECORD		
No. of duplicates[9]	d	d

[1] Number of hashed indexes to be stored in the same storage area.

[2] Assume the maximum size for the system record.

[3] A single unsigned byte is used to store the key length.

[4] The VARCHAR (or VARYING STRING) data type includes a 2-byte length column that is ignored for the index column length. The key value is space filled in the index. One byte (the 1 of K+1) is used to indicate null values.

[5] Duplicate count column (4) plus dbkey pointer (8).

[6] Size of each hash entry in the hash bucket is the sum of key size plus key length plus overhead per entry.

[7] An entry is any key value that maps to the same page, but is not a duplicate of an existing entry.

[8] Record type (4) plus overflow bucket dbkey pointer (8) and flags column (1)

[9] Number of duplicate records for the same value.

(continued on next page)

Table 8–1 (Cont.) Estimating the Number of Bytes per Entry Plus Overhead Bytes for Each Respective Index Record Type on a Data Page

Category	Bytes per Entry	Total
DUPLICATE NODE RECORD		
No. of entries/node[10]	10	
No. of duplicate nodes[11]	(d+5)/10=e	e
Overhead/node[12]	92	92
Total	e*92	e*92
GRAND TOTAL FOR THE HASHED INDEX		
	[((6*a)+4)] OR	[((10*a)+4)]+[(b*c)+13]+[e*92]

[10] The maximum number of duplicate entries that can fit in a duplicate node.

[11] The number of duplicate nodes rounded up to nearest whole number; if there are no duplicate records, then no duplicate nodes are created.

[12] Total overhead of one duplicate node.

- The record size

  Use the uncompressed size of user-stored data when you determine record size. The record size is one of the calculations you need to determine an adequate page size.

- A storage strategy

  Two basic strategies for storing records using a hashed index for which the PLACEMENT VIA INDEX clause has been specified in the table's storage map are as follows:

  – Store the records and the hashed bucket in the same mixed area.

    Using this approach, retrieval can be achieved in an average of one I/O operation. It is best for stable information, without excessive variance of duplicates.

  – Store the records and the hash bucket in separate mixed areas.

    This is the more conservative of the two approaches. Using this approach, the hash buckets are placed in one storage area, and the records in another. Retrieval can be achieved in an average of two I/O operations.

- SPAM information—thresholds and intervals
  - Thresholds

    If these are calculated incorrectly, the database will either have to search longer than necessary to find a free page to store the data, or it will fetch pages only to find that there is no room to store them. The second case can be seen by looking at the Performance Monitor Record Statistics screen.
  - Intervals

    Incorrect setting of the SPAM interval can result in additional I/O or increased page locking. A large interval tends to reduce I/O and increase page locking when multiple update users are accessing that portion of the storage area. A small SPAM interval reduces page locking problems (Performance Monitor Summary Locking Statistics screen) but increases the number of I/O operations required to locate free space for a record.

    The default page interval is 216 pages. If the storage area is large and page locking is not a problem, increasing this number can save I/O operations.

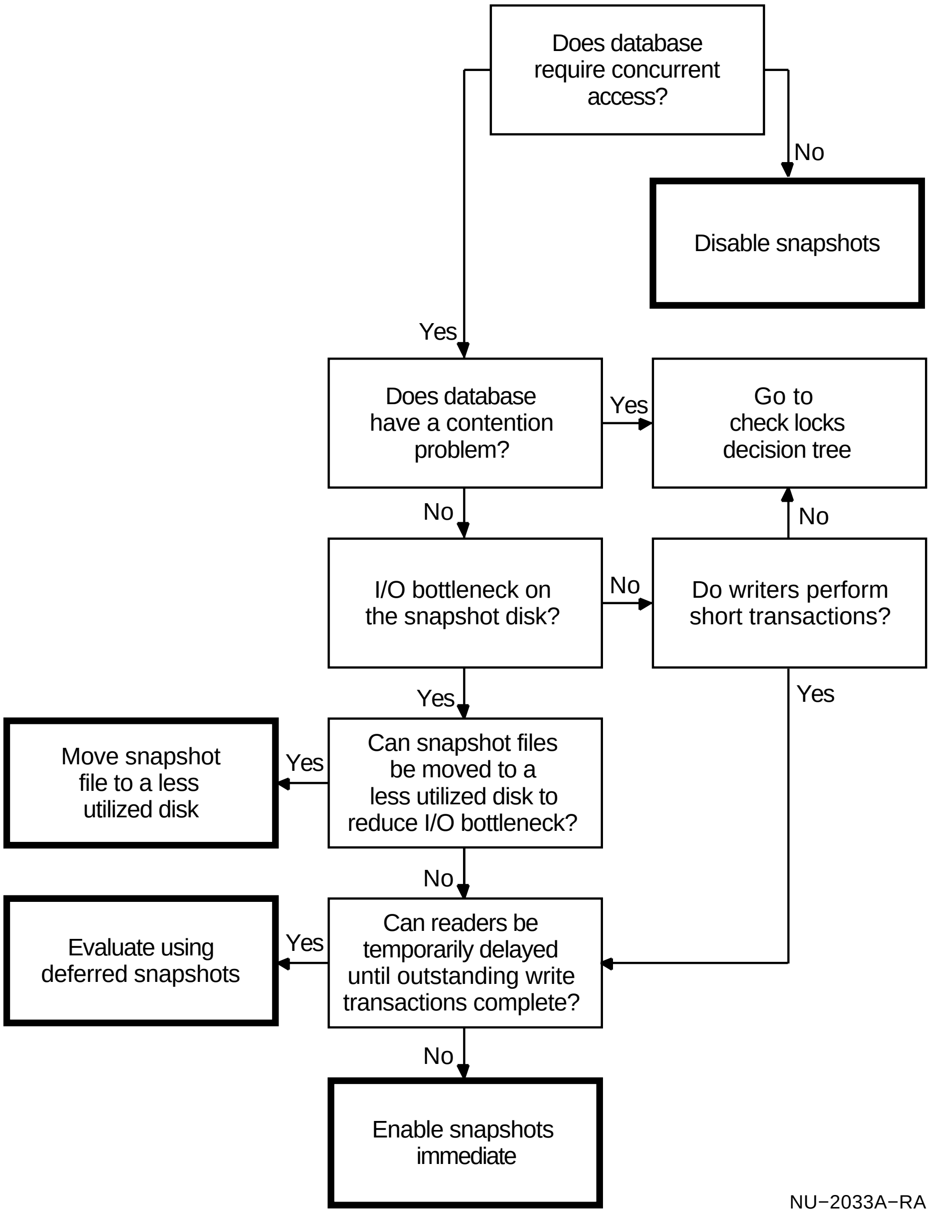
    Note that proper SPAM thresholds and intervals are also important to obtain good performance with sorted indexes.
- Key size
- An estimate of the number of unique keys and the number of duplicates
- An estimate of the total number of records associated with the hashed index

These final three items are needed to calculate the total page size.

8.1.3.6 Checking Snapshots

Snapshots are another area to consider when you attempt to reduce I/O (see Figure 8–16).

Figure 8–16 Decision Tree: Check Snapshots


Does database require concurrent access?
No
Disable snapshots
Yes
Does database have a contention problem?
Yes
Go to check locks decision tree
No
No
I/O bottleneck on the snapshot disk?
No
Do writers perform short transactions?
Yes
Yes
Move snapshot file to a less utilized disk
Yes
Can snapshot files be moved to a less utilized disk to reduce I/O bottleneck?
No
Evaluate using deferred snapshots
Yes
Can readers be temporarily delayed until outstanding write transactions complete?
No
Enable snapshots immediate
NU−2033A−RA

If the database is being used in a single-user application that does not require concurrent access to the information, then disable the snapshots. Do not incur the overhead of writing to the snapshot file if it is not necessary.

The default is that snapshot files are enabled immediate. This is probably the correct setting for snapshot files in an interactive, multiuser environment. When snapshots are enabled immediate, writers incur the overhead of updating the snapshot file every time they write to the database. When many users are accessing the data, performance will normally be optimal using immediate snapshots. This option trades additional I/O for reduced contention.

If snapshots are required and I/O resources are already limited on the disk where the storage area resides, consider moving the snapshot file to a less utilized device. If possible, place the .rda, .snp, .rdb, and .aij files so that each file type is on a separate disk.

Deferred snapshots can be used for those environments where it is acceptable for readers to be temporarily delayed. If your database is update-intensive with short transactions, and readers are not constantly using the database, you can benefit from deferred snapshots. Refer to Section 4.1.13 for additional information.

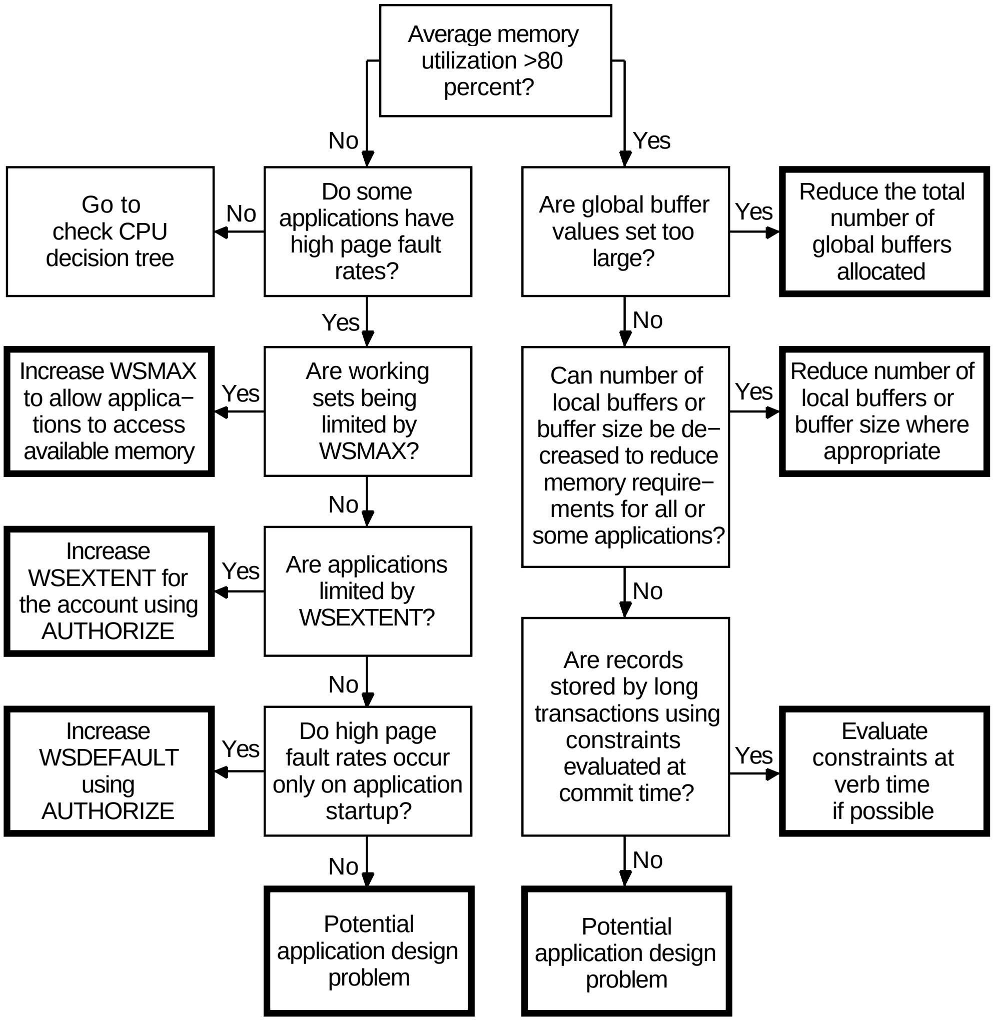
With deferred snapshots, read/write transactions write to the snapshot file only if a read-only transaction is in progress when the write transaction begins. The potential delay for the read-only transaction occurs when a write transaction is active when the read-only transaction starts. The read-only transaction must wait for all active write transactions to complete. New writers, who start a transaction after the read-only transaction begins, write before-images of the rows they are about to update to the .snp file.

The benefit of using snapshots enabled deferred is reduced I/O; if no read-only transactions are active, writers need not write to the snapshot file.

8.2 Analyzing Memory Resources

OpenVMS VAX OpenVMS Alpha Figure 8–17 will help you to identify and solve performance problems relating to memory resources.

Figure 8–17 Decision Tree: Check Memory



Average memory utilization >80 percent?
No
Yes
Go to check CPU decision tree
No
Do some applications have high page fault rates?
Are global buffer values set too large?
Yes
Reduce the total number of global buffers allocated
Yes
No
Increase WSMAX to allow applica– tions to access available memory
Yes
Are working sets being limited by WSMAX?
Can number of local buffers or buffer size be de– creased to reduce memory require– ments for all or some applications?
Yes
Reduce number of local buffers or buffer size where appropriate
No
Increase WSEXTENT for the account using AUTHORIZE
Yes
Are applications limited by WSEXTENT?
No
No
Are records stored by long transactions using constraints evaluated at commit time?
Increase WSDEFAULT using AUTHORIZE
Yes
Do high page fault rates occur only on application startup?
Yes
Evaluate constraints at verb time if possible
No
No
Potential application design problem
Potential application design problem

NU–2035A–RA

If memory is over 80 percent utilized, it is possible that some applications have oversized buffers or too many buffers. This causes the applications to compete for available memory resources, and can cause degradation due to an increase in system overhead.

If some of your applications have a high page fault rate, check to see if your working set extent limits how large your working set will grow. Use the SHOW PROCESS/ACCOUNTING command to see how large your peak working set size has become.

You can run the Authorize utility to see your current working set limits. If your peak working set size is the same as your WSEXTENT value, your process may have been limited by the current setting of that parameter. If memory is available, increasing WSEXTENT allows the process to access it. The working set range between WSQUOTA and WSEXTENT is used by the application only if it needs the memory and if the memory is available. The application can use up to WSQUOTA of working set without asking the system if it is available to use. WSDEFAULT is the amount of working set that a process has when it is created or when a user logs in to an account. If you see large page fault rates only when a process starts or when a user first logs in, consider increasing WSDEFAULT.

Another parameter to check if you have a large page fault rate and you feel that your application is not getting access to available memory is WSMAX in SYSGEN. WSMAX is the maximum working set size that any process on the system is allowed. WSMAX is frequently used to limit working set growth in a clustered environment that has a common UAF file and different amounts of memory resources on each node. However, if your value for WSEXTENT is larger than WSMAX, your process' peak working set will never extend beyond the value specified by WSMAX. In this situation, you might decide to increase the WSMAX value to a value larger than WSEXTENT.

```
$ RUN SYSGEN
SYSGEN>  USE ACTIVE
SYSGEN>  SHOW WSMAX
Parameter Name           Current   Default   Minimum   Maximum Unit  Dynamic
--------------           -------   -------   -------   ------- ----  -------
WSMAX                       8200      1024        60    100000 Pages
SYSGEN>  EXIT
$
```

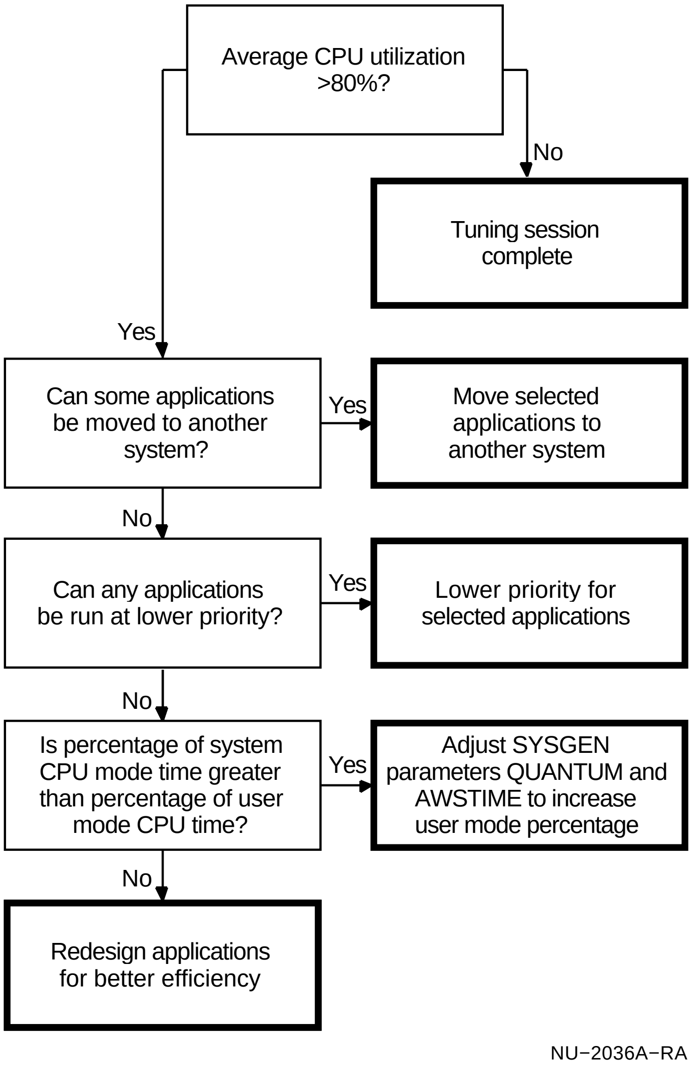
♦

8.3 Analyzing CPU Resources

OpenVMS VAX ≡ OpenVMS Alpha ≡

CPU resource limitations are difficult to address. Figure 8–18 identifies the steps you can follow to evaluate your resources. Options include moving some applications to another system or limiting the number of users that can access the machine. If your batch and interactive processing occur at the same priority, you can lower the priority on the batch activity to give interactive users first chance at the CPU resources. Carefully consider this option before you lower the priority of any write transactions. Resources required for your higher priority application could be held by the lower priority applications, leading to increased contention problems.

Figure 8–18 Decision Tree: Check CPU

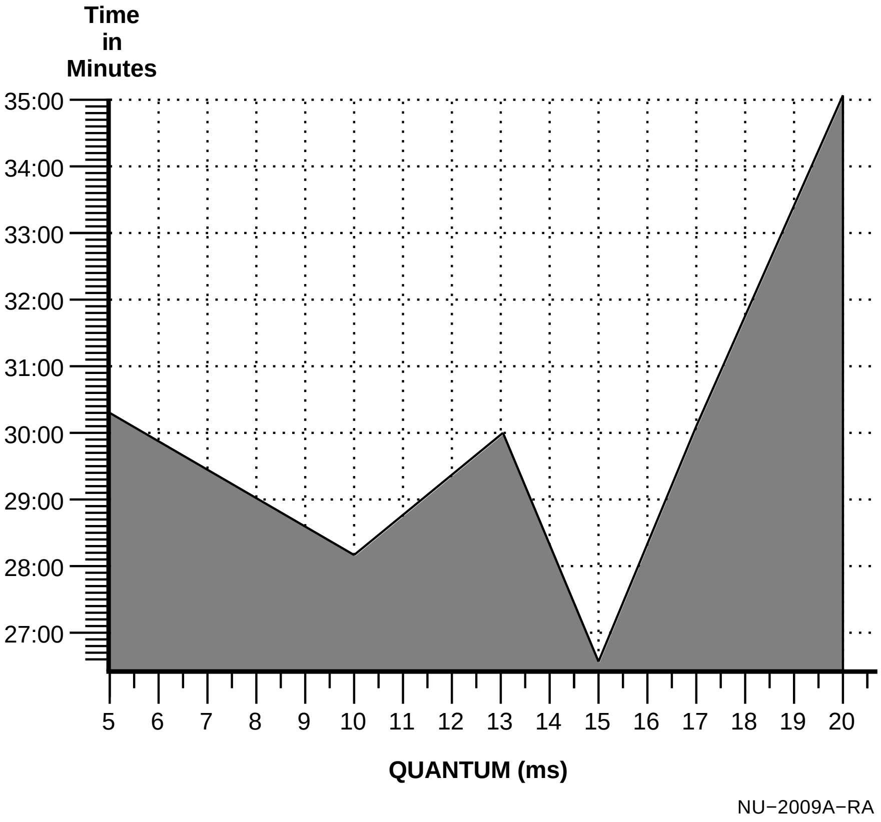

Average CPU utilization >80%?
No
Tuning session complete
Yes
Can some applications be moved to another system?
Yes
Move selected applications to another system
No
Can any applications be run at lower priority?
Yes
Lower priority for selected applications
No
Is percentage of system CPU mode time greater than percentage of user mode CPU time?
Yes
Adjust SYSGEN parameters QUANTUM and AWSTIME to increase user mode percentage
No
Redesign applications for better efficiency
NU−2036A−RA

Another option that you can sometimes use to help reduce a CPU resource limitation is to change the SYSGEN parameters QUANTUM and AWSTIME. QUANTUM is the CPU time given by the system for your process to use before the next process gets a turn. AWSTIME is the length of time the system waits before trying to adjust working sets. If you set QUANTUM lower than 10 milliseconds (ms), another special parameter, called IOTA, might be lowered from the default of 2 ms to 1 ms to keep the process from reaching QUANTUM end too soon. IOTA is the number of ms subtracted from the process' remaining QUANTUM at I/O completion to cause I/O-intensive processes to give up the CPU. These parameters are dynamic, so they can be changed and their effect measured immediately, without taking the system up and down. To use these parameters effectively, you need to consider your highest priority application, and determine if it is I/O or CPU-intensive (disk I/O, not buffered). I/O-intensive applications tend to benefit from smaller values of QUANTUM. CPU-intensive applications benefit from higher QUANTUM settings. These parameters cannot be changed on a per application basis. Because the changes affect all processes on the system, be careful when you change these parameters.

The resource savings from optimal setting of these parameters come from reduced operating system overhead. These savings can be used for user-related work. When QUANTUM is set lower than the default, a value of AWSTIME of 2 times QUANTUM is recommended. When QUANTUM is increased above the default, AWSTIME should be set the same as QUANTUM. In both cases, the system overhead from checking working set adjustments is reduced. For CPU-intensive applications, those applications that would be forced to wait for their next allocation of CPU time to complete may complete right away with a large QUANTUM. This improves response time and reduces the overhead required to temporarily save the program status for its next turn.

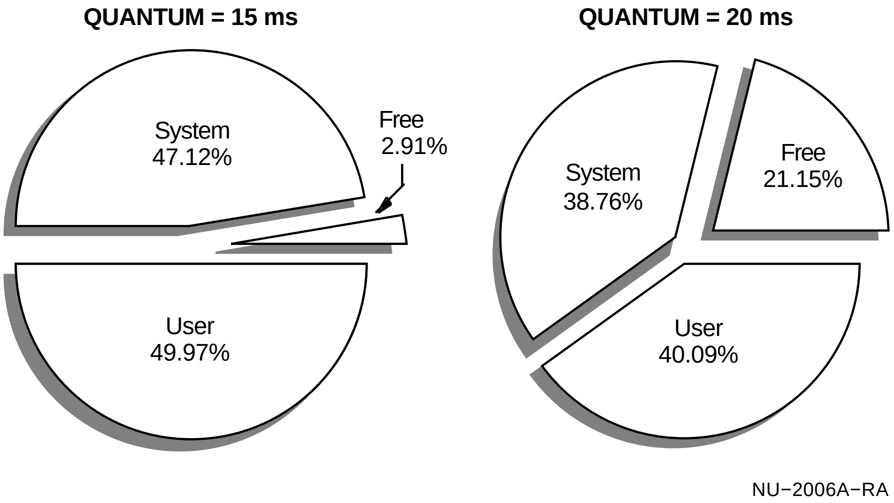
Graphs from a QUANTUM tuning study (Figure 8–19 and Figure 8–20) have been included to help illustrate this information. Figure 8–19 shows the effect that QUANTUM has on the target workload response time.

Figure 8–19 Effect of QUANTUM on Workload Response Time



These response time improvements occur with very little variation in CPU time. Also, memory and working sets remained the same for each iteration of the test. The slight variations in CPU usage can be attributed in part to the new automatic working set adjustment period for each interval. This caused the application working set to respond differently, which caused minor changes in direct I/O or page fault rate. Figure 8–20 compares CPU modes for QUANTUM settings of 15 and 20 ms.

Figure 8–20 Comparison of CPU Modes for QUANTUM Settings of 15 and 20 ms



♦

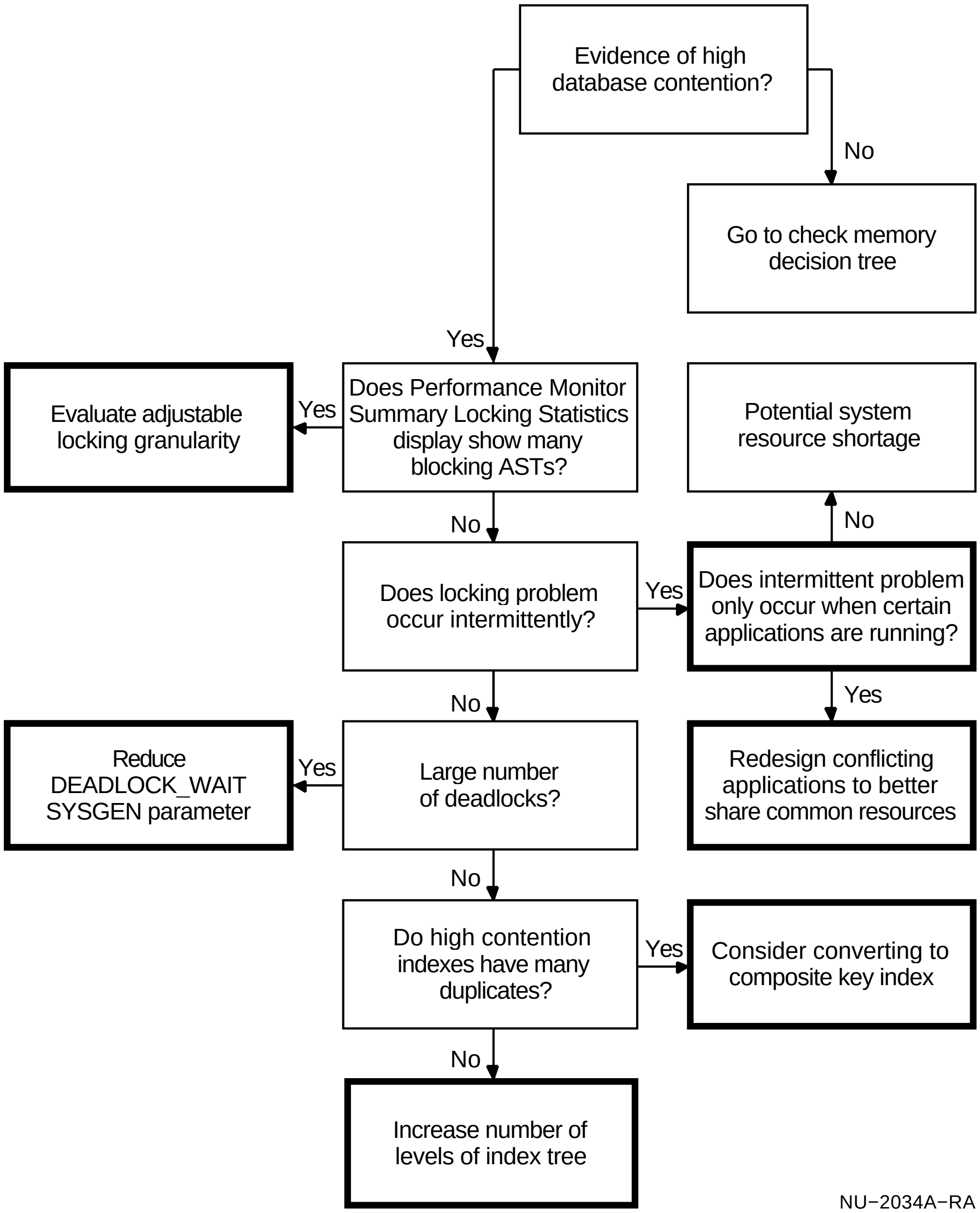
8.4 Analyzing Lock Resources

Locking can have a significant impact on database application response times. Locking is often the cause of intermittent performance problems where the application exhibits response time delays, yet CPU, I/O, and memory resources are still available.

There are many causes of locking problems. An application may be holding a shared resource with too restrictive a lock, or a resource may have too many users contending for it. These problems will limit the maximum database throughput that can be achieved.

Figure 8–21 summarizes the steps you can follow to check for locking problems.

Figure 8–21 Decision Tree: Check Locks


Evidence of high database contention?
No
Go to check memory decision tree
Yes
Does Performance Monitor Summary Locking Statistics display show many blocking ASTs?
Yes
Evaluate adjustable locking granularity
No
Does locking problem occur intermittently?
Yes
Does intermittent problem only occur when certain applications are running?
No
Potential system resource shortage
Yes
Redesign conflicting applications to better share common resources
No
Large number of deadlocks?
Yes
Reduce DEADLOCK_WAIT SYSGEN parameter
No
Do high contention indexes have many duplicates?
Yes
Consider converting to composite key index
No
Increase number of levels of index tree
NU−2034A−RA

If contention is evident in the database, for example, if lots of lock resources are being used up, or if many applications are stalled in the Performance Monitor Stall Messages screen, or if the Performance Monitor Summary Locking Statistics screen (Example 8–12) shows many blocking ASTs, then turning off adjustable lock granularity (ALG) should reduce the number of blocking ASTs by allowing Oracle Rdb to use the lowest level of locking immediately. (An increase in blocking ASTs may also be related to the carry-over lock optimization; see Section 3.8.3.2 for details.) Refer to Section 3.8.5 for information on ALG. Examine the DECLARE TRANSACTION and SET TRANSACTION statements in your application code to ensure that they allow maximum concurrency. If your transaction is only reading a table, set the transaction to read-only and the access mode to shared read (specify the SQL SET TRANSACTION READ ONLY RESERVING table-name FOR SHARED READ statement for the transaction). In SQL, the default is read/write and shared write. Thus, if your application uses the default, it will hold more protective locks on the resources than are necessary if the transaction is a read-only transaction. Symptoms of this problem include increased contention and lower concurrency than expected.

Example 8–12 Performance Monitor Summary Locking Statistics Screen

```
Node: MYNODE         Oracle Rdb V7.0-00 Performance Monitor 30-MAY-1996 14:59:39
Rate: 3.00 Seconds         Summary Locking Statistics       Elapsed: 00:30:14.24
Page: 1 of 1     USER18:[PRODUCTION.DATABASE]PRODUCT_DB.RDB;1       Mode: Online
--------------------------------------------------------------------------------
statistic.........      rate.per.second............ total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

locks requested            78       52       36.8        59013         115.3
 rqsts not queued           1        0        0.2          349           0.7
 rqsts stalled              2        0        0.2          390           0.8
 rqst timeouts              0        0        0.0            0           0.0
 rqst deadlocks             0        0        0.0            0           0.0
locks promoted             60       47       14.9        23840          46.6
 proms not queued           0        0        0.0           29           0.1
 proms stalled              0        0        0.0           20           0.0
 prom timeouts              0        0        0.0            0           0.0
 prom deadlocks             0        0        0.0            0           0.0
locks demoted             101       78       22.7        36417          71.1
locks released            190       35       36.4        58324         113.9
blocking ASTs               4        0        0.4          615           1.2
stall time x100             5        0        1.0         1630           3.2
invalid lock block          0        0        0.0            0           0.0
--------------------------------------------------------------------------------
Exit Graph Help Menu Options Pause Reset Set_rate Time_plot Unreset Write X_plot
```

If the locking problems are intermittent, an application may be accessing the database sequentially instead of using an index. When an application starts a read/write transaction in shared write mode and the access is sequential, the shared write lock is promoted to protected write (PW). This has the effect of locking the whole table, and could account for intermittent locking problems. To see if an application is using the expected index, use the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter. Setting the name to S results in the display of the retrieval strategy, as shown in the following example:

```
$ SQL
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SELECT JOB_TITLE FROM JOBS WHERE JOB_CODE = "VPSD";
   .
   .
   .
Conjunct        Get     Retrieval sequentially of relation JOBS
 JOB_TITLE
 Vice President
1 row selected
SQL>EXIT
$
```

For detailed information on using the RDMS$DEBUG_FLAGS logical name and the RDB_DEBUG_FLAGS configuration parameter, see Appendix C.

Another area that may be degrading performance is excessive deadlocks. Deadlocks can be caused by applications or by poor application design.

OpenVMS VAX OpenVMS Alpha

If your application is prone to deadlocks, you can adjust the SYSGEN parameter DEADLOCK_WAIT to detect the situation more quickly. The default value is 10 seconds. Lower the parameter if the environment is deadlock prone.

By lowering the parameter, deadlocks are detected more quickly and applications are delayed for less time. The cost of this is increased overhead to search the lock timeout queue. If you get very few deadlocks, increasing this parameter causes true deadlock situations to be undetected longer, but reduces the overhead of searching for deadlocks that exist only rarely. The more locks that are allocated, the more time it takes to search the queue and the greater the overhead to search for a deadlock condition.

```
$ RUN SYSGEN
SYSGEN>  USE ACTIVE
SYSGEN>  SHOW DEADLOCK_WAIT
Parameter Name            Current   Default   Minimum   Maximum Unit  Dynamic
--------------            -------   -------   -------   ------- ----  -------
DEADLOCK_WAIT                  10        10         0        -1 Seconds     D
SYSGEN> EXIT
```

Note that lowering the DEADLOCK_WAIT parameter only helps you to detect deadlocks more quickly. ♦

If many deadlocks are encountered in a database, you can fix some of them by changing the database or applications. Some of the changes you could make to the database to fix deadlocks include reducing the size of sorted index nodes or converting a sorted index to a hashed index. Changes that you could make to applications include using update only cursors or using a restricted mode (such as protected write mode) for transactions (using a restricted mode may fix the deadlock, but could cause contention problems while the transaction is executing). In some cases, deadlocks are unavoidable.

A

Oracle Rdb Logical Names and Configuration Parameters

This appendix describes Oracle Rdb logical names and configuration parameters, and explains when and how to use them to improve database performance.

The logical names and configuration parameters that are specific to the Hot Standby option are not documented in this appendix. See the *Oracle Rdb7 and Oracle CODASYL DBMS: Guide to Hot Standby Databases* for information about the logical names and configuration parameters that are specific to the Hot Standby option.

A.1 RDB$CHARACTER_SET

OpenVMS VAX OpenVMS Alpha

You can define an alternate character set for use by Oracle Rdb. Valid alternate character sets include the following:

- DEC_KANJI, Japan
- DEC_HANZI, PRC-China
- DEC_HANGUL, Korea
- DEC_HANYU, Taiwan

Example A–1 sets the character set to Japanese.

Example A–1 Using the RDB$CHARACTER_SET Logical Name

```
$ DEFINE RDB$CHARACTER_SET DEC_KANJI
```

The RDB$CHARACTER_SET logical name will be deprecated in a future release. ♦

A.2 RDB$LIBRARY and RDB_LIBRARY

Specifies a protected library that you can use to store external routine images, such as external functions. Oracle Corporation recommends that you manage public or sensitive external routine images using a protected library that is referenced by the logical name RDB$LIBRARY or the configuration parameter RDB_LIBRARY.

OpenVMS VAX OpenVMS Alpha

You should define RDB$LIBRARY as an executive mode logical name in the system logical name table. If the external routine image is located in the protected area, you can ensure that the desired image is used by specifying the RDB$LIBRARY logical name with an explicit file name in the LOCATION clause plus the WITH SYSTEM LOGICAL_NAME TRANSLATION clause in a CREATE FUNCTION statement. ♦

A.3 RDB$RDBSHR_EVENT_FLAGS

OpenVMS VAX OpenVMS Alpha

On OpenVMS, RDB$SHARE is assigned the following event flags at startup:

- Stall event flag
- Common event flag
- Remote read event flag
- Remote write event flag

These event flags are used for internal DECdtm processing. The common event flag has a default value of 31. The other event flag values are assigned by the LIB$GET_EF system service by default.

You can use the RDB$RDBSHR_EVENT_FLAGS logical name to override the default event flag values. You may wish to do this if you are using event flags for other purposes and you wish to restrict Oracle Rdb event flags to a particular range or set of values.

Specify the logical name value as a string containing four integer values in the range 0 to 63. Enclose the string in quotes, and separate the integers with commas. These values are assigned to event flags in the same order the event flags are shown in the previous list.

If an error occurs in translating the RDB$RDBSHR_EVENT_FLAGS logical name, or if the specified event flag value is out of range, Oracle Rdb uses the default event flag values assigned by the LIB$GET_EF system service. Example A–2 shows how to use the RDB$RDBSHR_EVENT_FLAGS logical name.

Example A–2 Using the RDB$RDBSHR_EVENT_FLAGS Logical Name

```
$ DEFINE RDB$RDBSHR_EVENT_FLAGS "63,62,61,60"
```

The RDB$RDBSHR_EVENT_FLAGS logical name is translated using the LMN$DCL_LOGICAL logical table. ♦

A.4 RDB$REMOTE_BUFFER_SIZE and SQL_NETWORK_BUFFER_SIZE

By default, the buffer size of network transfers is 4096 bytes. You can use the RDB$REMOTE_BUFFER_SIZE logical name or the SQL_NETWORK_BUFFER_SIZE configuration parameter to change the default buffer size of network transfers. The minimum value is 500 bytes and the maximum value is limited only by your system's resources and quota limits. If you specify a value of less than 500 bytes, Oracle Rdb uses the default value of 4096 bytes.

It may be advantageous to increase your network buffer size before running an application if you transfer large data blocks into or out of the database. Increasing the buffer size reduces the number of network I/O operations used when large transfers are made.

Digital UNIX

On Digital UNIX, include the line shown in Example A–3 in your configuration file to set the buffer size to 10,000 bytes.

Example A–3 Using the SQL_NETWORK_BUFFER_SIZE Configuration Parameter

```
SQL_NETWORK_BUFFER_SIZE 10000
```

♦

You can set the network buffer size on one or both of the client and server nodes in their respective configuration files. If these values are different for a particular client and server combination, the smaller of the two values is used.

OpenVMS VAX OpenVMS Alpha

The RDB$REMOTE_BUFFER_SIZE logical name is translated using the LNM$DCL_LOGICAL logical table. ♦

A.5 RDB$REMOTE_MULTIPLEX_OFF and SQL_NETWORK_NUMBER_ATTACHES

The RDB$REMOTE_MULTIPLEX_OFF logical name and the SQL_NETWORK_NUMBER_ATTACHES configuration parameter control the number of remote server processes used for multiple remote database accesses to the same node.

If you define any value for the RDB$REMOTE_MULTIPLEX_OFF logical name or set the SQL_NETWORK_NUMBER_ATTACHES configuration parameter to the value of 1, each of your remote database accesses will require its own RDB_SERVER process on the remote node. If you assign a higher value to the SQL_NETWORK_NUMBER_ATTACHES configuration parameter, up to that number of your remote database accesses to a particular node will share a single RDB_SERVER process on the remote node. The default value for SQL_NETWORK_NUMBER_ATTACHES is 10.

Digital UNIX

On Digital UNIX, include the line shown in Example A–4 in your configuration file to increase the network attaches to 20.

Example A–4 Using the SQL_NETWORK_NUMBER_ATTACHES Configuration Parameter

```
SQL_NETWORK_NUMBER_ATTACHES 20
```

♦

OpenVMS VAX OpenVMS Alpha

The RDB$REMOTE_MULTIPLEX_OFF logical name is translated using the LNM$DCL_LOGICAL logical table. ♦

A.6 RDB$ROUTINES and RDB_ROUTINES

Specifies the location of an external routine image. If you do not specify a location clause in a CREATE FUNCTION statement, or if you specify the DEFAULT LOCATION clause, SQL uses the RDB$ROUTINES logical name or the RDB_ROUTINES configuration parameter as the default image location.

A.7 RDBVMS$CREATE_DB and RDB_CREATE_DB

You can restrict the creation of databases by defining the logical name RDBVMS$CREATE_DB or the configuration parameter RDB_CREATE_DB.

OpenVMS VAX OpenVMS Alpha

On OpenVMS, you must also define a rights identifier of the same name. ♦

> **Note**
>
> When you define RDBVMS$CREATE_DB or RDB_CREATE_DB, other installed Oracle and third-party products will not be able to use Oracle Rdb to create Oracle Rdb databases. Therefore, you must deassign the logical name or remove the parameter from the configuration file whenever users of such products need to create an Oracle Rdb database.

OpenVMS VAX OpenVMS Alpha

On OpenVMS, to restrict the creation of Oracle Rdb databases, first define the logical name RDBVMS$CREATE_DB as shown in Example A–5.

Example A–5 Using the RDBVMS$CREATE_DB Logical Name

```
$ DEFINE/SYSTEM/EXECUTIVE RDBVMS$CREATE_DB RESTRICT_DB
```

Next, use the rights identifier, RDBVMS$CREATE_DB, to control which users can create databases using the SQL CREATE DATABASE statement. For more information, see the *Oracle Rdb7 Guide to Database Design and Definition*. ♦

A.8 RDM$BIND_ABS_LOG_FILE and RDB_BIND_ABS_LOG_FILE

You can use the RDM$BIND_ABS_LOG_FILE logical name or the RDB_BIND_ABS_LOG_FILE configuration parameter to define a file name for the after-image journal backup server (ABS) log file.

OpenVMS VAX OpenVMS Alpha

You must define this logical name in the LNM$SYSTEM_TABLE table. ♦

A.9 RDM$BIND_ABS_OVERWRITE_ALLOWED and RDB_BIND_ABS_OVERWRITE_ALLOWED

You can use the logical name RDM$BIND_ABS_OVERWRITE_ALLOWED or the configuration parameter RDB_BIND_ABS_OVERWRITE_ALLOWED to indicate whether the after-image journal backup server (ABS) resets overwritten AIJ journals. The default value 0 indicates that the ABS cannot reset overwritten journals, while the value 1 indicates that the ABS can reset overwritten journals.

A.10 RDM$BIND_ABS_OVERWRITE_IMMEDIATE and RDB_BIND_ABS_OVERWRITE_IMMEDIATE

You can use the logical name RDM$BIND_ABS_OVERWRITE_IMMEDIATE or the configuration parameter RDB_BIND_ABS_OVERWRITE_IMMEDIATE to indicate whether journals should be immediately reset if RDM$BIND_ABS_OVERWRITE_ALLOWED or RDB_BIND_ABS_OVERWRITE_ALLOWED is enabled.

The default value 0 indicates that AIJ journals should not be immediately reset. The value 1 indicates that AIJ journals should be immediately reset.

A.11 RDM$BIND_ABS_QUIET_POINT and RDB_BIND_ABS_QUIET_POINT

You can use the logical name RDM$BIND_ABS_QUIET_POINT or the configuration parameter RDB_BIND_ABS_QUIET_POINT to indicate whether the after-image journal backup server (ABS) will perform a quiet-point after-image journal backup.

The default value 0 indicates that a no-quiet-point backup will be performed while 1 indicates that a quiet-point backup will be performed.

You should define the logical name and configuration parameter on all nodes that access the database.

OpenVMS VAX ☰ OpenVMS Alpha ☰

You should define RDM$BIND_ABS_QUIET_POINT as an executive mode logical name in the system logical name table as shown in the following example:

```
$ DEFINE/SYSTEM/EXEC RDM$BIND_ABS_QUIET_POINT 1
```

♦

Note that the value you assign to RDM$BIND_ABS_QUIET_POINT or RDB_BIND_ABS_QUIET_POINT is in effect for all databases on the current node.

A.12 RDM$BIND_ABW_ENABLED and RDB_BIND_ABW_ENABLED

You can disable or enable asynchronous batch-write operations with the RDM$BIND_ABW_ENABLED logical name or the RDB_BIND_ABW_ENABLED configuration parameter. The default value 1 indicates that asynchronous batch-write operations are enabled, and the value 0 indicates that they are disabled.

Digital UNIX

On Digital UNIX, include the line shown in Example A–6 in your configuration file to disable asynchronous batch-write operations.

Example A–6 Using the RDB_BIND_ABW_ENABLED Configuration Parameter

```
RDB_BIND_ABW_ENABLED 0
```

♦

Asynchronous batch-write operations are enabled by default.

For more information on asynchronous batch-write operations, see Section 3.2.5.

A.13 RDM$BIND_AIJ_CHECK_CONTROL_RECS and RDB_BIND_AIJ_CHECK_CONTROL_RECS

The RDM$BIND_AIJ_CHECK_CONTROL_RECS logical name and the RDB_BIND_AIJ_CHECK_CONTROL_RECS configuration parameter indicate whether to check for control records during AIJ cache formatting. The default value 1 indicates that Oracle Rdb will check for control records, and the value 0 indicates that Oracle Rdb will not check for control records.

A.14 RDM$BIND_AIJ_EMERGENCY_DIR and RDB_BIND_AIJ_EMERGENCY_DIR

You can use the RDM$BIND_AIJ_EMERGENCY_DIR logical name or the RDB_BIND_AIJ_EMERGENCY_DIR configuration parameter to specify the location of the emergency AIJ journal. This logical name or configuration parameter should specify only the device and directory where the emergency AIJ journal is to be created. Note that, if defined, the RDM$BIND_AIJ_EMERGENCY_DIR logical name or the RDB_BIND_AIJ_EMERGENCY_DIR configuration parameter applies to all databases on the current node.

OpenVMS VAX OpenVMS Alpha

On OpenVMS, the logical name must not contain any non-system concealed logical definitions and must reside in the LNM$SYSTEM_TABLE logical name table. ♦

A.15 RDM$BIND_AIJ_IO_MAX and RDB_BIND_AIJ_IO_MAX

The RDM$BIND_AIJ_IO_MAX logical name and the RDB_BIND_AIJ_IO_MAX configuration parameter allow you to override the maximum AIJ group commit I/O buffer size. The default buffer size is 127 blocks.

A.16 RDM$BIND_AIJ_IO_MIN and RDB_BIND_AIJ_IO_MIN

The RDM$BIND_AIJ_IO_MIN logical name and the RDB_BIND_AIJ_IO_MIN configuration parameter allow you to override the minimum AIJ group commit I/O buffer size. The default buffer size is 8 blocks.

A.17 RDM$BIND_AIJ_STALL and RDB_BIND_AIJ_STALL

The RDM$BIND_AIJ_STALL logical name and the RDB_BIND_AIJ_STALL configuration parameter define the amount of time, expressed in milliseconds, that a transaction waits after submitting its commit record to the after-image journal (.aij) log file. This wait time permits a larger number of transactions in the group commit operation.

Based on various benchmarks, an optimal wait time of 50 milliseconds has been established as the default value. However, the true optimal value depends on the particular application. If an application is not update-intensive, a smaller value may improve performance slightly. If an application is update-intensive, a larger value may result in slightly better performance.

Note

Oracle Corporation recommends that you do not change the default value unless it is absolutely necessary.

The default value is 50 milliseconds. The minimum value is 0 milliseconds, and the maximum value is 1000 milliseconds (1 second).

A.18 RDM$BIND_AIJ_SWITCH_GLOBAL_CKPT and RDB_BIND_AIJ_SWITCH_GLOBAL_CKPT

You can use the RDM$BIND_AIJ_SWITCH_GLOBAL_CKPT logical name or the RDB_BIND_AIJ_SWITCH_GLOBAL_CKPT configuration parameter to indicate whether to perform a global checkpoint after an AIJ switch-over has occurred. The default value 1 indicates that a global checkpoint will be performed, and the value 0 indicates that a global checkpoint will not be performed.

A.19 RDM$BIND_ALS_CREATE_AIJ and RDB_BIND_ALS_CREATE_AIJ

You can use the RDM$BIND_ALS_CREATE_AIJ logical name or the RDB_BIND_ALS_CREATE_AIJ configuration parameter to indicate whether the ALS server is to create an emergency AIJ journal if the AIJ switch-over operation enters the suspended state.

A database enters the "AIJ suspended" state if the AIJ switch-over operation cannot complete because there are no available AIJ journals. During this state, the DBA can add new AIJ journals or perform database backups, but all other AIJ-related activities are temporarily suspended until an AIJ journal becomes available.

During the AIJ suspended period, any DBR invocation causes the database to be shut down. This is required because the DBR always writes either a commit or rollback record to the AIJ journal. Note that even a DBR invoked for a read-only transaction causes the database to be shut down.

The default value 0 indicates that the ALS should not create an AIJ journal and the value 1 indicates that the ALS should attempt to create an AIJ journal. On OpenVMS, the logical name must reside in the LNM$SYSTEM_TABLE logical name table.

When the RDM$BIND_ALS_CREATE_AIJ logical is set to the value 1, the ALS attempts to create an emergency AIJ journal using the previous AIJ journal as a template. This means that the emergency AIJ journal is created in the same directory, and with the same allocation, as the journal being switched *from*. If there is no adequate disk space, or any other error occurs, the database enters the "AIJ suspended" state and the DBA must resolve the situation.

Warning

The emergency AIJ journal is not a temporary AIJ journal. Do not delete it from DCL. You should only delete the emergency AIJ journal with SQL or RMU syntax. Manually deleting the emergency AIJ journal will cause your database to be shut down.

The ALS will notify the DBA through the operator notification facility that an emergency AIJ journal has been created. Furthermore, the output from the RMU Dump Header command will identify any AIJ journal created by the ALS server process. The Performance Monitor also highlights any identified emergency AIJ journal.

An emergency AIJ journal has the following characteristics:

- An emergency AIJ journal is a normal AIJ journal in all respects. It is created by the ALS process to avoid the AIJ switch-over suspension state. DBR invocations due to application process failure during the AIJ journal creation do not cause the database to shut down.
- The name of the emergency AIJ journal is EMERGENCY_XXX where XXX is a series of 16 characters used to create a unique name.
- The creation of the emergency AIJ journal is not journaled. This means that the creation of an emergency AIJ journal is not possible while Hot Standby database replication is active.
- There is no way to remove the emergency status of an emergency AIJ journal.

A.20 RDM$BIND_APF_DEPTH and RDB_BIND_APF_DEPTH

Specifies the number (depth) of buffers for Oracle Rdb to asynchronously prefetch for a process.

OpenVMS VAX OpenVMS Alpha

Example A–7 shows how to specify that Oracle Rdb prefetch 5 buffers asynchronously for a process.

Example A–7 Using the RDM$BIND_APF_DEPTH Logical Name

```
$ DEFINE RDM$BIND_APF_DEPTH 5
```

♦

See Section 3.2.4 for more information on using the Oracle Rdb asynchronous prefetch capability.

A.21 RDM$BIND_APF_ENABLED and RDB_BIND_APF_ENABLED

The logical name RDM$BIND_APF_ENABLED and the configuration parameter RDB_BIND_APF_ENABLED allow you to disable or enable asynchronous prefetch operations. The default value 1 indicates that asynchronous prefetch operations are enabled and the value 0 indicates that they are disabled.

Digital UNIX

Include the line shown in Example A–8 in your configuration file to disable asynchronous prefetch operations.

Example A–8 Using the RDB_BIND_APF_ENABLED Configuration Parameter

```
RDB_BIND_APF_ENABLED 0
```

♦

Asynchronous prefetch operations are enabled by default.

For more information on asynchronous prefetch operations, see Section 3.2.4.

A.22 RDM$BIND_BATCH_MAX and RDB_BIND_BATCH_MAX

The RDM$BIND_BATCH_MAX logical name and the RDB_BIND_BATCH_MAX configuration parameter define the number of live data page cache buffers that are written to the database as part of a batch-write operation or asynchronous batch-write operation. A batch-write operation occurs when a new cache buffer is needed but no buffers are available. To rectify this situation, Oracle Rdb writes one or more of the existing cache buffers to the database, making them candidates for replacement. Asynchronous batch-write operations occur when the number of clean buffers at the end of a process' least recently used queue of buffers for replacement is less than the number of clean buffers specified for the process with the RDM$BIND_CLEAN_BUF_CNT logical name or the RDB_BIND_CLEAN_BUF_CNT configuration parameter.

When the RDM$BIND_BATCH_MAX logical name or the RDB_BIND_BATCH_MAX configuration parameter is undefined, batch-write operations result in large I/O bursts that cause unpredictable system performance. By setting this logical name or configuration parameter to a value smaller than the number of cache buffers allocated to the user, you can control the size of the I/O burst. This results in more predictable and uniform I/O behavior for the system.

The optimal value of this logical name and configuration parameter depends on several factors, including the number of cache buffers allocated per user, the number of storage areas in the database, the type and speed of the disk drives, the CPU load, the application workload, and so on. Setting the value too small may result in excessive batch-write operations. Setting the value too large may result in large I/O burst patterns.

The value of this logical name and configuration parameter does not affect the size of the batch-write operation for snapshot pages or the size of the batch used for commit or checkpoint operations.

The default value of the RDM$BIND_BATCH_MAX logical name and the RDB_BIND_BATCH_MAX configuration parameter is the number of cache buffers allocated to the user. This is also the maximum value, even if the number specified is larger than the number of buffers used. The minimum value is 1.

See Section 4.1.2.2 for more information on how Oracle Rdb determines the size of a user's allocate set.

A.23 RDM$BIND_BUFFERS and RDB_BIND_BUFFERS

You can use the RDM$BIND_BUFFERS logical name or the RDB_BIND_BUFFERS configuration parameter to change the number of buffers Oracle Rdb allocates for each database user at run time. By default, Oracle Rdb allocates the value specified by the NUMBER OF BUFFERS IS parameter. The behavior of RDM$BIND_BUFFERS and RDB_BIND_BUFFERS depends on whether you are using local or global buffers.

- If you have local buffers enabled (the default), you can specify a value between 2 and 524288 for RDM$BIND_BUFFERS and RDB_BIND_BUFFERS.
- If you have global buffers enabled, the number of buffers you specify with RDM$BIND_BUFFERS or RDB_BIND_BUFFERS cannot exceed the value set by the USER LIMIT IS parameter. The USER LIMIT IS value defines the maximum number of global buffers a process can allocate from the global buffer pool and overrides an RDM$BIND_BUFFERS or RDB_BIND_BUFFERS value that exceeds that value.

Section 4.1.2 describes how and when to use the RDM$BIND_BUFFERS logical name and the RDB_BIND_BUFFERS configuration parameter with both local and global buffers.

The RDM$BIND_BUFFERS logical name and the RDB_BIND_BUFFERS configuration parameter can be a powerful tool for tuning specific applications. For example, you can define the number of buffers to exceed the default for an initial load program to facilitate database loading and sequential database searches. Also, you can use RDM$BIND_BUFFERS or RDB_BIND_BUFFERS to allocate more buffers for batch programs that run during off-peak hours, but still retain the default number of buffers for users during normal working hours.

OpenVMS VAX OpenVMS Alpha

Example A–9 allocates 100 buffers to the process that defines the logical name.

Example A–9 Using the RDM$BIND_BUFFERS Logical Name

```
$ DEFINE RDM$BIND_BUFFERS 100
```

If you have the required privileges, you can define RDM$BIND_BUFFERS as a group or system logical name to affect a larger group of processes. ♦

A.24 RDM$BIND_BUFOBJ_ENABLED

OpenVMS Alpha

Oracle Rdb includes a performance enhancement feature for databases accessed from OpenVMS Alpha systems. This feature uses the OpenVMS buffer object feature to lock Oracle Rdb local buffers into physical memory. Locking Oracle Rdb local buffers into memory increases SMP parallelism and scaling and improves I/O performance by eliminating OpenVMS overhead.

To utilize this feature you must:

- Run Oracle Rdb V6.1 or higher along with OpenVMS Alpha Version 6.1 or higher on an Alpha uniprocessor or an SMP system.
- Define the logical name RDM$BIND_BUFOBJ_ENABLED to be any value.
- Increase the OpenVMS user quota BYTLM by the number of bytes in your local buffer pool. For example, if the number of local buffers is 1000, and the buffer size is 8 blocks, determine the value of the BYTLM parameter using the following formula:

$$1000 \ * \ (8 \ * \ 512) \ = \ 4,096,000 \ bytes$$

Do not use this feature on systems that are memory constrained. OpenVMS pages that are defined as a buffer object can be paged and swapped, but must remain resident in physical memory. This physical memory becomes unavailable for use by any other process until the image is terminated.

The Oracle Rdb utilization of OpenVMS buffer objects is not available on databases with global buffers enabled. ♦

A.25 RDM$BIND_CBL_ENABLED and RDB_BIND_CBL_ENABLED

The RDM$BIND_CBL_ENABLED logical name and the RDB_BIND_CBL_ENABLED configuration parameter indicate whether coarse buffer locking is enabled. The default value 0 indicates that coarse buffer locking is disabled, and the value 1 indicates that it is enabled.

A.26 RDM$BIND_CKPT_BLOCKS and RDB_BIND_CKPT_BLOCKS

You can use the RDM$BIND_CKPT_BLOCKS logical name or the RDB_BIND_CKPT_BLOCKS configuration parameter to specify the number of AIJ blocks after which a checkpoint will occur. The default value is 0 blocks.

A.27 RDM$BIND_CKPT_TIME and RDB_BIND_CKPT_TIME

You can use the RDM$BIND_CKPT_TIME logical name or the RDB_BIND_CKPT_TIME configuration parameter to specify the amount of time, in seconds, after which a checkpoint will occur. The default value is 0.

A.28 RDM$BIND_CKPT_TRANS_INTERVAL and RDB_BIND_CKPT_TRANS_INTERVAL

You can use the RDM$BIND_CKPT_TRANS_INTERVAL logical name or the RDB_BIND_CKPT_TRANS_INTERVAL configuration parameter to define a process-specific checkpoint interval value. By default, if you have fast commit processing enabled, a process checkpoints when the AIJ block size limit is reached or the time interval limit is exceeded, whichever occurs first. The RDM$BIND_CKPT_TRANS_INTERVAL logical name and the RDB_BIND_CKPT_TRANS_INTERVAL configuration parameter use the number of transactions as the checkpoint trigger. Thus if you define RDM$BIND_CKPT_TRANS_INTERVAL or RDB_BIND_CKPT_TRANS_INTERVAL to be 10, the process on which the logical name or configuration parameter is defined checkpoints after committing 10 transactions *if* the transaction limit is reached before the block size or time limits are reached. See Section 4.1.5.2 for more information.

A.29 RDM$BIND_CLEAN_BUF_CNT and RDB_BIND_CLEAN_BUF_CNT

You can use the RDM$BIND_CLEAN_BUF_CNT logical name or the RDB_BIND_CLEAN_BUF_CNT configuration parameter to specify the number of clean buffers to be maintained at the end of a process' least recently used queue of buffers for replacement. This logical name and configuration parameter are used as part of the asynchronous batch-write feature, which is designed to reduce the number of I/O stalls experienced by a process. The default value for RDM$BIND_CLEAN_BUF_CNT and RDB_BIND_CLEAN_BUF_CNT is 5, which means that five clean buffers will be maintained at the end of a process' least recently used queue of buffers for replacement.

The RDM$BIND_BATCH_MAX logical name and the RDB_BIND_BATCH_MAX configuration parameter are also used with the asynchronous batch-write feature. See Section A.22 for more information on RDM$BIND_BATCH_MAX and RDB_BIND_BATCH_MAX.

See Section 3.2.5 for information on the asynchronous batch-write feature.

A.30 RDM$BIND_COMMIT_STALL and RDB_BIND_COMMIT_STALL

The RDM$BIND_COMMIT_STALL logical name and the RDB_BIND_COMMIT_STALL configuration parameter define the amount of time, expressed in milliseconds, that a transaction waits after attempting to become the group commit process. This wait time permits a larger number of transactions in the group commit operation.

Based on various benchmarks, an optimal wait time of 50 milliseconds has been established as the default value. However, the true optimal value depends on the particular application. If an application is not update-intensive, a smaller value may improve performance slightly. If an application is update-intensive, a larger value may result in slightly better performance.

Note

Oracle Corporation recommends that you do not change the default value unless it is absolutely necessary.

The default value is 50 milliseconds. The minimum value is 0 milliseconds, and the maximum value is 1000 milliseconds (1 second).

A.31 RDM$BIND_DAPF_DEPTH_BUF_CNT and RDB_BIND_DAPF_DEPTH_BUF_CNT

The logical name RDM$BIND_DAPF_DEPTH_BUF_CNT and the configuration parameter RDB_BIND_DAPF_DEPTH_BUF_CNT allow you to specify the number of buffers to prefetch from the physical area. The default is half the number of buffers defined for the database user.

A.32 RDM$BIND_DAPF_ENABLED and RDB_BIND_DAPF_ENABLED

The logical name RDM$BIND_DAPF_ENABLED and the configuration parameter RDB_BIND_DAPF_ENABLED allow you to disable detected asynchronous prefetch (DAPF) read operations. The DAPF feature predicts the next page to be read for a physical area if the physical area is to be read sequentially. If the page read matches the previous prediction, then Oracle Rdb asynchronously prefetches pages prior to the actual (sequential) read request.

The default value 1 indicates that detected asynchronous prefetch operations are enabled and the value 0 indicates that they are disabled.

Digital UNIX

Include the line shown in Example A–10 in your configuration file to disable asynchronous prefetch operations.

Example A–10 Using the RDB_BIND_DAPF_ENABLED Configuration Parameter

```
RDB_BIND_DAPF_ENABLED 0
```

♦

The DAPF feature improves performance primarily in mixed-format areas and when you are:

- Inserting a large number of records in which the records would get stored on successive pages
- Performing an RMU Verify command, or other operations that access the PIO (or page) layer of Oracle Rdb by passing the DIO (or record) layer
- Walking adjacent sorted index leaf nodes from left to right when nodes are in adjacent clumps

Detected asynchronous prefetch read operations are enabled by default.

A.33 RDM$BIND_DAPF_START_BUF_CNT and RDB_BIND_DAPF_START_BUF_CNT

The logical name RDM$BIND_DAPF_START_BUF_CNT and the configuration parameter RDB_BIND_DAPF_START_BUF_CNT allow you to specify the number of buffers to be accessed sequentially from the physical area before detected asynchronous prefetch read operations start.

A.34 RDM$BIND_HRL_ENABLED and RDM_BIND_HRL_ENABLED

The RDM$BIND_HRL_ENABLED logical name and the RDM_BIND_HRL_ENABLED configuration parameter indicate whether hold retrieval locks are enabled. The default value 0 indicates that hold retrieval locks are disabled and the value 1 indicates that they are enabled.

A.35 RDM$BIND_LOCK_TIMEOUT_INTERVAL and RDB_BIND_LOCK_TIMEOUT_INTERVAL

You can specify a default wait interval by defining the RDM$BIND_LOCK_TIMEOUT_INTERVAL logical name or the RDB_BIND_LOCK_TIMEOUT_INTERVAL configuration parameter.

Digital UNIX

Example A–11 shows how to define the configuration parameter on Digital UNIX.

Example A–11 Using the RDB_BIND_LOCK_TIMEOUT_INTERVAL Configuration Parameter

```
RDB_BIND_LOCK_TIMEOUT_INTERVAL 15
```

Example A–11 specifies a wait interval of 15 seconds. Note that the wait interval is expressed in seconds, but that the time period is approximate. The amount of time you specify for the wait interval depends on your application. ♦

OpenVMS VAX OpenVMS Alpha

However, as a general guideline, use a value greater than the value specified in the SYSGEN parameter DEADLOCK_WAIT. See Section 4.4.2 for information on the DEADLOCK_WAIT parameter. ♦

To specify an application-specific wait interval, use the WAIT clause of the SQL SET TRANSACTION statement. The interval specified by the WAIT clause supersedes the interval specified by the logical name or configuration parameter.

You can set a database-wide default lock timeout interval, which functions as an upper limit on the amount of time that can be set with the WAIT clause or with the RDM$BIND_LOCK_TIMEOUT_INTERVAL logical name or the RDB_BIND_LOCK_TIMEOUT_INTERVAL configuration parameter. This database-wide lock timeout interval is set with the LOCK TIMEOUT INTERVAL IS *n* SECONDS parameter of the SQL CREATE or ALTER DATABASE statement.

See the *Oracle Rdb7 Guide to Distributed Transactions* and the *Oracle Rdb7 SQL Reference Manual* for more information.

A.36 RDM$BIND_MAX_DBR_COUNT and RDB_BIND_MAX_DBR_COUNT

The RDM$BIND_MAX_DBR_COUNT logical name and the RDB_BIND_MAX_DBR_COUNT configuration parameter define the maximum number of database recovery (DBR) processes to be simultaneously invoked by the database monitor. This logical name and configuration parameter apply only to databases that do not have global buffers enabled. Databases that utilize global buffers have only one recovery process started at a time.

OpenVMS VAX OpenVMS Alpha On OpenVMS, this logical name must be defined in the LNM$SYSTEM_TABLE logical name table. The following example limits the maximum number of recovery processes to 20:

```
$ DEFINE/SYSTEM RDM$BIND_MAX_DBR_COUNT 20
```

♦

A.37 RDM$BIND_OPTIMIZE_AIJ_RECLEN and RDB_BIND_OPTIMIZE_AIJ_RECLEN

You can improve the performance of the RMU Optimize After_Journal command by defining the RDM$BIND_OPTIMIZE_AIJ_RECLEN logical name or the RDB_BIND_OPTIMIZE_AIJ_RECLEN configuration parameter.

In general, Oracle Corporation recommends the following guidelines to improve the overall after-image journal optimization performance:

- Always use the RDMS$BIND_SORT_WORKFILES logical name or the RDB_BIND_SORT_WORKFILES configuration parameter to specify the number of work files you wish to use.
- Always use the SORTWORKn logical names to specify the file names of temporary work files on an unused, preferably fast, device.
- Never put two or more work files on the same device.
- Use fewer work files if the work file devices have lots of free space. Use more work files if available free space is limited.
- Do not use the Trace qualifier with the RMU Optimize After_Journal command if you do not need to trace the output. The trace output greatly increases the number of buffered and direct I/O operations and the overall elapsed time.
- Use the Log qualifier to obtain the OPTRECLEN message.

- Use the output of the OPTRECLEN message plus 10 percent as the value of the RDM$BIND_OPTIMIZE_AIJ_RECLEN logical name or the RDB_BIND_OPTIMIZE_AIJ_RECLEN configuration parameter; the minimum value is 512 and the maximum value is 4096.
- Do not optimize an .aij file containing many DDL records or .aij records greater than the RDM$BIND_OPTIMIZE_AIJ_RECLEN or RDB_BIND_OPTIMIZE_AIJ_RECLEN value; these types of records require an immediate sort and flush operation, which is extremely expensive and generates larger output files. The exact number of records depends largely on the application and the overall size of the input after-image journal.

A.38 RDM$BIND_RCACHE_INSERT_ENABLED and RDB_BIND_RCACHE_INSERT_ENABLED

You can use the RDM$BIND_RCACHE_INSERT_ENABLED logical name or the RDM$BIND_RCACHE_INSERT_ENABLED configuration parameter to indicate whether or not rows can be inserted into the row cache. The default value 1 indicates that rows can be inserted into the cache, and the value 0 indicates that rows cannot be inserted into the cache.

A.39 RDM$BIND_RCACHE_RCRL_COUNT and RDB_BIND_RCACHE_RCRL_COUNT

You can use the RDM$BIND_RCACHE_RCRL_COUNT logical name or the RDB_BIND_RCACHE_RCRL_COUNT configuration parameter to specify the number of row cache slots to reserve. Oracle Rdb reserves 20 row cache slots by default.

A.40 RDM$BIND_RCS_BATCH_COUNT and RDB_BIND_RCS_BATCH_COUNT

The RDM$BIND_RCS_BATCH_COUNT logical name and the RDB_BIND_RCS_BATCH_COUNT configuration parameter define the number of rows that the row cache server (RCS) process sweeps in a single batch.

The default value is 3000 rows. The minimum value is 1 row and the maximum value is 1,000,000 rows.

OpenVMS VAX OpenVMS Alpha On OpenVMS, this logical name must be defined in the LNM$SYSTEM_TABLE logical name table. ♦

A.41 RDM$BIND_RCS_CHECKPOINT and RDB_BIND_RCS_CHECKPOINT

You can use the RDM$BIND_RCS_CHECKPOINT logical name or the RDB_BIND_RCS_CHECKPOINT configuration parameter to direct the row cache server (RCS) to perform a checkpoint. The default value 1 indicates a checkpoint is performed, and the value 0 indicates that a checkpoint is not performed. When the RCS process performs a checkpoint, it writes all of the marked cold records in the row caches to their respective backing store files.

A.42 RDM$BIND_RCS_CKPT_BUFFER_CNT and RDB_BIND_RCS_CKPT_BUFFER_CNT

You can use the RDM$BIND_RCS_CKPT_BUFFER_CNT logical name or the RDB_BIND_RCS_CKPT_BUFFER_CNT configuration parameter to indicate the number of buffers to be examined as a single batch by the row cache server (RCS) process during a checkpoint operation. The default value for the buffer count is 15 buffers.

A.43 RDM$BIND_RCS_LOG_FILE and RDB_BIND_RCS_LOG_FILE

You can use the RDM$BIND_RCS_LOG_FILE logical name or the RDB_BIND_RCS_LOG_FILE configuration parameter to define a file name for the row cache server (RCS) log file.

OpenVMS VAX OpenVMS Alpha You must define this logical name in the LNM$SYSTEM_TABLE table. ♦

A.44 RDM$BIND_RCS_MAX_COLD and RDB_BIND_RCS_MAX_COLD

You can use the RDM$BIND_RCS_MAX_COLD logical name or the RDB_BIND_RCS_MAX_COLD configuration parameter to specify the number of marked records above which the row cache server (RCS) sweep starts. When the RCS process performs a sweep, it writes all of the marked cold records in the row caches to their respective storage areas.

A.45 RDM$BIND_RCS_MIN_COLD and RDB_BIND_RCS_MIN_COLD

You can use the RDM$BIND_RCS_MIN_COLD logical name or the RDB_BIND_RCS_MIN_COLD configuration parameter to specify the number of unmarked records below which the row cache server (RCS) sweep completes. When the RCS process performs a sweep, it writes all of the marked cold records in the row caches to their respective storage areas.

A.46 RDM$BIND_RCS_SWEEP_INTERVAL and RDB_BIND_RCS_SWEEP_INTERVAL

You can use the RDM$BIND_RCS_SWEEP_INTERVAL logical name or the RDB_BIND_RCS_SWEEP_INTERVAL configuration parameter to specify the amount of time, in minutes, between sweeps. The default sweep interval is 1 minute.

A.47 RDM$BIND_READY_AREA_SERIALLY and RDB_BIND_READY_AREA_SERIALLY

The RDM$BIND_READY_AREA_SERIALLY logical name and the RDB_BIND_READY_AREA_SERIALLY configuration parameter cause Oracle Rdb to grant lock requests for logical and physical areas in the order that the lock requests were made.

The lock manager has two queues, the WAIT and CONVERSION queues, to grant lock requests. When a process makes a lock request, one of two things happens. The process either gets the lock immediately in the requested mode, or (if another process has the lock already in an incompatible mode) the process is put on the WAIT queue, where it is granted an NL lock to the resource (regardless of the mode the process was requesting for the resource). Once the process gets the lock in NL mode, it is moved from the WAIT queue to the CONVERSION queue, where it seeks to get the NL mode lock converted to a lock in the desired (originally requested) mode. On the CONVERSION queue, lock requests compatible with the currently granted mode are given preference over other incompatible lock requests on the queue (that is, a lock request on the CONVERSION queue that is compatible with the currently granted mode is granted before incompatible lock requests that have been waiting longer on the CONVERSION queue). Lock requests in the CONVERSION queue are always given precedence over lock requests in the WAIT queue.

Moving a process lock request quickly from the WAIT queue to the CONVERSION queue reduces the possibility of lock starvation. Lock starvation is the term used to describe the situation where a process will never be granted a lock in the requested mode. For example, suppose many users on the CONVERSION queue kept readying and unreadying areas. If an incompatible lock request on the WAIT queue was not moved quickly to the CONVERSION queue, the process on the WAIT queue could starve for the lock (because lock requests in the CONVERSION queue are given precedence over lock requests in the WAIT queue).

Lock requests on the CONVERSION queue that are compatible with the currently granted mode are given preference over incompatible lock requests on the CONVERSION queue. This means that if there are many compatible lock requests on the CONVERSION queue and very few incompatible lock requests, the incompatible requests could become starved for the lock. To prevent the lock starvation that can occur in this situation, define RDM$BIND_READY_AREA_SERIALLY or RDB_BIND_READY_AREA_SERIALLY as 1; when this is done, Oracle Rdb will force a serial ordering on the CONVERSION queue. Thus, when RDM$BIND_READY_AREA_SERIALLY or RDB_BIND_READY_AREA_SERIALLY is defined as 1, Oracle Rdb grants lock requests on the CONVERSION queue for logical and physical areas in the order that the lock requests were made.

OpenVMS VAX OpenVMS Alpha

Example A–12 shows how to define the RDM$BIND_READY_AREA_SERIALLY logical name to prevent lock starvation.

Example A–12 Using the RDM$BIND_READY_AREA_SERIALLY Logical Name

```
$ DEFINE/SYSTEM/EXECUTIVE RDM$BIND_READY_AREA_SERIALLY 1
```

♦

A.48 RDM$BIND_RUJ_ALLOC_BLKCNT and RDB_BIND_RUJ_ALLOC_BLKCNT

You can use the RDM$BIND_RUJ_ALLOC_BLKCNT logical name or the RDB_BIND_RUJ_ALLOC_BLKCNT configuration parameter to define the initial size, in blocks, of the .ruj file. The default .ruj file size is 127 blocks.

A.49 RDM$BIND_RUJ_EXTEND_BLKCNT and RDB_BIND_RUJ_EXTEND_BLKCNT

You can use the RDM$BIND_RUJ_EXTEND_BLKCNT logical name or the RDB_BIND_RUJ_EXTEND_BLKCNT configuration parameter to pre-extend .ruj files for each process using a database. For example, you can define the new block count to be a value between 1 and 65535 blocks or accept the default of 127 blocks.

Digital UNIX

On Digital UNIX, include the line shown in Example A–13 in your configuration file to pre-extend the .ruj file.

Example A–13 Using the RDB_BIND_RUJ_EXTEND_BLKCNT Configuration Parameter

```
RDB_BIND_RUJ_EXTEND_BLKCNT 1000
```

♦

A.50 RDM$BIND_SNAP_QUIET_POINT and RDB_BIND_SNAP_QUIET_POINT

The logical name RDM$BIND_SNAP_QUIET_POINT and the configuration parameter RDB_BIND_SNAP_QUIET_POINT indicate whether snapshot transactions hold the quiet-point lock and stall database or after-image journal backups from operating.

The default value 1 indicates read-only transactions will continue to hold the quiet-point lock. If the value is 0, the quiet-point lock is released prior to a read-only transaction.

The process' modified buffers are flushed to the appropriate database files prior to releasing the quiet-point lock.

A.51 RDM$BIND_STATS_AIJ_ARBS_PER_IO and RDB_BIND_STATS_AIJ_ARBS_PER_IO

The RDM$BIND_STATS_AIJ_ARBS_PER_IO logical name and the RDB_BIND_STATS_AIJ_ARBS_PER_IO configuration parameter allow you to override the default value of AIJ request blocks per AIJ I/O. The default is 2 blocks.

You can also set this threshold from the configuration submenu in the Performance Monitor AIJ Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.52 RDM$BIND_STATS_AIJ_BKGRD_ARB_RATIO and RDB_BIND_STATS_AIJ_BKGRD_ARB_RATIO

The RDM$BIND_STATS_AIJ_BKGRD_ARB_RATIO logical name and the RDB_BIND_STATS_AIJ_BKGRD_ARB_RATIO configuration parameter allow you to override the default value for the background AIJ request block threshold. The default value is 50.

You can also set this threshold from the configuration submenu in the Performance Monitor AIJ Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.53 RDM$BIND_STATS_AIJ_BLKS_PER_IO and RDB_BIND_STATS_AIJ_BLKS_PER_IO

The RDM$BIND_STATS_AIJ_BLKS_PER_IO logical name and the RDB_BIND_STATS_AIJ_BLKS_PER_IO configuration parameter allow you to override the default value of blocks per AIJ I/O. The default value is 2.

You can also set this threshold from the configuration submenu in the Performance Monitor AIJ Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.54 RDM$BIND_STATS_AIJ_SEC_TO_EXTEND and RDB_BIND_STATS_AIJ_SEC_TO_EXTEND

The RDM$BIND_STATS_AIJ_SEC_TO_EXTEND logical name and the RDB_BIND_STATS_AIJ_SEC_TO_EXTEND configuration parameter allow you to override the default value of seconds to AIJ extend. The default value is 60.

You can also set this threshold from the configuration submenu in the Performance Monitor AIJ Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.55 RDM$BIND_STATS_BTR_FETCH_DUP_RATIO and RDB_BIND_STATS_BTR_FETCH_DUP_RATIO

The RDM$BIND_STATS_BTR_FETCH_DUP_RATIO logical name and the RDB_BIND_STATS_BTR_FETCH_DUP_RATIO configuration parameter allow you to override the default value of the B-tree duplicate fetch threshold. The default threshold is 15.

You can also set this threshold from the configuration submenu in the Performance Monitor Index Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.56 RDM$BIND_STATS_BTR_LEF_FETCH_RATIO and RDB_BIND_STATS_BTR_LEF_FETCH_RATIO

The RDM$BIND_STATS_BTR_LEF_FETCH_RATIO logical name and the RDB_BIND_STATS_BTR_LEF_FETCH_RATIO configuration parameter allow you to override the default value of the B-tree leaf node fetch threshold. The default threshold is 25.

You can also set this threshold from the configuration submenu in the Performance Monitor Index Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.57 RDM$BIND_STATS_DBR_RATIO and RDB_BIND_STATS_DBR_RATIO

The RDM$BIND_STATS_DBR_RATIO logical name and the RDB_BIND_STATS_DBR_RATIO configuration parameter allow you to override the default value of the DBR invocation threshold. The default threshold is 15.

You can also set this threshold from the configuration submenu in the Performance Monitor RUJ Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.58 RDM$BIND_STATS_ENABLED and RDB_BIND_STATS_ENABLED

You can disable the writing of database statistics for a process with the logical name RDM$BIND_STATS_ENABLED or the configuration parameter RDB_BIND_STATS_ENABLED. When database statistics are disabled for a process, the Performance Monitor shows zeros in the fields for each of the display screens for that process. If you want to disable the writing of statistics for all processes on a node, you must define the logical name RDM$BIND_STATS_ENABLED or the configuration parameter RDB_BIND_STATS_ENABLED for each process on that node. By default, the writing of database statistics is enabled for each process on a node; the value is set to 1. Disabling statistics is useful for static, performance-critical applications that have been previously tuned and do not need the information provided by the Performance Monitor. To disable the writing of database statistics for a process, define the RDM$BIND_STATS_ENABLED logical name or the RDB_BIND_STATS_ENABLED configuration parameter to 0.

OpenVMS VAX OpenVMS Alpha

Example A–14 shows how to disable database statistics for a process.

Example A–14 Using the RDM$BIND_STATS_ENABLED Logical Name

```
$ DEFINE RDM$BIND_STATS_ENABLED 0
```

♦

To enable the writing of database statistics for a process in which the collection of database statistics is disabled, define the RDM$BIND_STATS_ENABLED logical name or the RDB_BIND_STATS_ENABLED configuration parameter to

1 or deassign the logical name or remove the parameter from the configuration file.

OpenVMS VAX OpenVMS Alpha Example A–15 shows how to enable database statistics for a process.

Example A–15 Using the RDM$BIND_STATS_ENABLED Logical Name

```
$ DEFINE RDM$BIND_STATS_ENABLED 1
$
$ ! Or you can deassign the logical name
$
$ DEASSIGN RDM$BIND_STATS_ENABLED
```

♦

A.59 RDM$BIND_STATS_FULL_BACKUP_INTRVL and RDB_BIND_STATS_FULL_BACKUP_INTRVL

The RDM$BIND_STATS_FULL_BACKUP_INTRVL logical name and the RDB_BIND_STATS_FULL_BACKUP_INTRVL configuration parameter allow you to override the full database backup threshold. The default threshold is 6.

You can also set this threshold from the configuration submenu in the Performance Monitor RUJ Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.60 RDM$BIND_STATS_GB_IO_SAVED_RATIO and RDB_BIND_STATS_GB_IO_SAVED_RATIO

The RDM$BIND_STATS_GB_IO_SAVED_RATIO logical name and the RDB_BIND_STATS_GB_IO_SAVED_RATIO configuration parameter allow you to override the GB IO-saved default threshold. The default threshold is 85.

You can also set the global buffer IO-saved threshold from the configuration submenu in the Performance Monitor Buffer Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.61 RDM$BIND_STATS_GB_POOL_HIT_RATIO and RDB_BIND_STATS_GB_POOL_HIT_RATIO

The RDM$BIND_STATS_GB_POOL_HIT_RATIO logical name and the RDB_BIND_STATS_GB_POOL_HIT_RATIO configuration parameter allow you to override the GB pool hit default threshold. The default threshold is 85.

You can also set the global buffer pool hit threshold from the configuration submenu in the Performance Monitor Buffer Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.62 RDM$BIND_STATS_LB_PAGE_HIT_RATIO and RDB_BIND_STATS_LB_PAGE_HIT_RATIO

The RDM$BIND_STATS_LB_PAGE_HIT_RATIO logical name and the RDB_BIND_STATS_LB_PAGE_HIT_RATIO configuration parameter allow you to override the LB/AS page hit default threshold. The default is 75.

You can also set the local buffer pool hit threshold from the configuration submenu in the Performance Monitor Buffer Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.63 RDM$BIND_STATS_MAX_HASH_QUE_LEN and RDB_BIND_STATS_MAX_HASH_QUE_LEN

The RDM$BIND_STATS_MAX_HASH_QUE_LEN logical name and the RDB_BIND_STATS_MAX_HASH_QUE_LEN configuration parameter allow you to override the hash table queue length default threshold. The default threshold is 2 rows.

You can also set the hash table queue length threshold from the configuration submenu in the Performance Monitor Transaction Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.64 RDM$BIND_STATS_MAX_LOCK_STALL and RDB_BIND_STATS_MAX_LOCK_STALL

The RDM$BIND_STATS_MAX_LOCK_STALL logical name and the RDB_BIND_STATS_MAX_LOCK_STALL configuration parameter allow you to override the lock stall default threshold. The default threshold is 2 seconds.

You can also set this threshold from the configuration submenu in the Performance Monitor Locking Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.65 RDM$BIND_STATS_MAX_TX_DURATION and RDB_BIND_STATS_MAX_TX_DURATION

The RDM$BIND_STATS_MAX_TX_DURATION logical name and the RDB_BIND_STATS_MAX_TX_DURATION configuration parameter allow you to override the transaction duration default threshold. The default value is 15.

You can also set the transaction duration threshold from the configuration submenu in the Performance Monitor Transaction Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.66 RDM$BIND_STATS_PAGES_CHECKED_RATIO and RDB_BIND_STATS_PAGES_CHECKED_RATIO

The RDM$BIND_STATS_PAGES_CHECKED_RATIO logical name and the RDB_BIND_STATS_PAGES_CHECKED_RATIO configuration parameter allow you to override the pages checked default threshold. The default threshold is 10 pages.

You can also set this threshold from the configuration submenu in the Performance Monitor Record Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.67 RDM$BIND_STATS_RECS_FETCHED_RATIO and RDB_BIND_STATS_RECS_FETCHED_RATIO

The RDM$BIND_STATS_RECS_FETCHED_RATIO logical name and the RDB_BIND_STATS_RECS_FETCHED_RATIO configuration parameter allow you to override the records fetched default threshold. The default threshold is 20 records.

You can also set this threshold from the configuration submenu in the Performance Monitor Record Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.68 RDM$BIND_STATS_RECS_STORED_RATIO and RDB_BIND_STATS_RECS_STORED_RATIO

The RDM$BIND_STATS_RECS_STORED_RATIO logical name and the RDB_BIND_STATS_RECS_STORED_RATIO configuration parameter allow you to override the records stored default threshold. The default threshold is 20 records.

You can also set this threshold from the configuration submenu in the Performance Monitor Record Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.69 RDM$BIND_STATS_RUJ_SYNC_IO_RATIO and RDB_BIND_STATS_RUJ_SYNC_IO_RATIO

The RDM$BIND_STATS_RUJ_SYNC_IO_RATIO logical name and the RDB_BIND_STATS_RUJ_SYNC_IO_RATIO configuration parameter allow you to override the synchronous RUJ I/O default threshold. The default threshold is 10.

You can also set this threshold from the configuration submenu in the Performance Monitor RUJ Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.70 RDM$BIND_STATS_VERB_SUCCESS_RATIO and RDB_BIND_STATS_VERB_SUCCESS_RATIO

The RDM$BIND_STATS_VERB_SUCCESS_RATIO logical name and the RDB_BIND_STATS_VERB_SUCCESS_RATIO configuration parameter allow you to override the verb success default threshold. The default threshold is 25.

You can also set the verb success threshold from the configuration submenu in the Performance Monitor Transaction Analysis screen. See Section 2.2.16 for more information on the Performance Monitor Online Analysis facility.

A.71 RDM$BIND_SYSTEM_BUFFERS_ENABLED

OpenVMS Alpha

You can use the logical name RDM$BIND_SYSTEM_BUFFERS_ENABLED, located in the LNM$SYSTEM_TABLE logical table, to specify whether or not system space global sections are used. The RDM$BIND_SYSTEM_BUFFERS_ENABLED logical name must be defined on *all* nodes that use system space global sections. This logical name is not used as a database parameter.

The RDM$BIND_SYSTEM_BUFFERS_ENABLED logical name has three values:

0	Indicates use of process global sections. This is the default value.
1	Indicates use of system space global sections, if possible. If this is not possible, then process global sections are used.
2	Indicates use of system space global sections. If available system space is inadequate, then the database open operation will fail.

The RMU Show Users command indicates whether system space global sections are enabled, and shows the active values for the global buffer parameters. ♦

A.72 RDM$BIND_TSN_INTERVAL and RDB_BIND_TSN_INTERVAL

The RDM$BIND_TSN_INTERVAL logical name and the RDB_BIND_TSN_INTERVAL configuration parameter allow you to specify the number of transactions to be allocated as a single batch when journal optimization is enabled.

A.73 RDM$BIND_VM_SEGMENT and RDB_BIND_VM_SEGMENT

You can use the RDM$BIND_VM_SEGMENT logical name or the RDB_BIND_VM_SEGMENT configuration parameter to prevent memory fragmentation and allow applications that would otherwise run out of virtual memory to execute correctly. Fragmentation occurs when, after some number of virtual memory allocations and deallocations, unallocated virtual memory exists as a collection of noncontiguous bytes (known as fragments). At some point, it is possible that none of the memory fragments are large enough to satisfy the next request for virtual memory. This logical name and configuration parameter are not for general use, because preserving larger block sizes can cause problems for some applications.

Unlike RDMS$BIND_WORK_VM and RDB_BIND_WORK_VM, which allocate a user-specified number of virtual memory bytes (restricted only by the amount of memory available on your system), RDM$BIND_VM_SEGMENT and RDB_BIND_VM_SEGMENT allocate whatever number of *contiguous* bytes are necessary to avoid memory fragmentation. Depending on the size of the available virtual memory blocks, applications may receive more virtual memory than requested. Defining RDM$BIND_VM_SEGMENT or RDB_BIND_VM_SEGMENT to be 1 enables this feature; defining it to be 0 disables it.

Digital UNIX

On Digital UNIX, include the line shown in Example A–16 in your configuration file to enable the parameter.

Example A–16 Using the RDB_BIND_VM_SEGMENT Configuration Parameter

```
RDB_BIND_VM_SEGMENT 1
```

♦

You should enable RDM$BIND_VM_SEGMENT or RDB_BIND_VM_SEGMENT either on the system experiencing the problem, or in some other appropriate place, such as the LOGIN.COM file or the configuration file of the user experiencing the fragmentation.

A.74 RDM$BUGCHECK_DIR and RDB_BUGCHECK_DIR

You can use the RDM$BUGCHECK_DIR logical name or the RDB_BUGCHECK_DIR configuration parameter to redirect the location of bugcheck files from the default directory to another location. This can be useful when the default directory does not have enough space for bugcheck files.

OpenVMS VAX OpenVMS Alpha

On OpenVMS, when the bugcheck directory is defined, bugcheck dump files are written to the device and directory pointed to by the RDM$BUGCHECK_DIR logical name, rather than to the user's top-level directory, which is the default behavior on OpenVMS. ♦

Digital UNIX

On Digital UNIX, when the bugcheck directory is defined, bugcheck dump files are written to the device and directory pointed to by the RDB_BUGCHECK_DIR configuration parameter, rather than to the directory where the database exists, which is the default behavior on Digital UNIX. ♦

When users have reached their disk quotas in their default directory due to a bugcheck dump, and if the RDM$BUGCHECK_DIR logical name or the RDB_BUGCHECK_DIR configuration parameter is not specified to another device, the bugcheck dump overflows to the KOD$TT file. In this case, a DBR process is created with an output device of KOD$TT. Also, note that bugcheck dump files are written to the system disk and if the system disk becomes full, the overflow might also end up in KOD$TT. Generally, nothing is written to KOD$TT and an examination of the KOD$TT file will show that it is empty.

OpenVMS VAX OpenVMS Alpha

DBR bugcheck dump files by default are written to the SYS$SYSTEM directory; however, when you define the RDM$BUGCHECK_DIR logical name, DBR bugchecks will also be written to the new location specified by this logical name. ♦

As with any file creation, the user must have read and write access to the bugcheck directory for the bugcheck dump file to be written successfully.

Digital UNIX

Example A–17 shows how to redirect the location of the bugcheck file.

Example A–17 Using the RDB_BUGCHECK_DIR Configuration Parameter

```
RDB_BUGCHECK_DIR /usr/tmp/bugcheck
```

♦

OpenVMS VAX OpenVMS Alpha

If the logical name you supply translates to the null device (NL:), then you disable the bugcheck output. However, this defeats the purpose of the feature; disabling the bugcheck output does not fix the problem that causes the bugcheck dump. See the *Oracle Rdb7 Guide to Database Maintenance* for more information. ♦

A.75 RDM$BUGCHECK_IGNORE_FLAGS and RDB_BUGCHECK_IGNORE_FLAGS

You can use the RDM$BUGCHECK_IGNORE_FLAGS logical name or the RDB_BUGCHECK_IGNORE_FLAGS configuration parameter to reduce the size of bugcheck dump files created by Oracle Rdb.

OpenVMS VAX OpenVMS Alpha

For example, to prevent the dumping of locking information and page dumps for all Oracle Rdb users on OpenVMS, issue the following command:

```
$ DEFINE/SYSTEM RDM$BUGCHECK_IGNORE_FLAGS "LP"
```

In this example the “L” stands for “Locking” and the “P” stands for “Pages”. ♦

The complete list of available flags is in Table A–1.

Table A–1 RDM$BUGCHECK_IGNORE_FLAGS and RDB_BUGCHECK_IGNORE_FLAGS

Flag	Description
C	Disables dumping client-specific information. For Oracle Rdb this is information such as request BLR, request (REQ) blocks, and generated code. Disabling the dumping of this information substantially limits the possibility of diagnosing problems such as optimizer problems, internally generated coding problems, and area mapping problems. Typically this portion of a bugcheck dump is not exceptionally large.

(continued on next page)

Table A–1 (Cont.) RDM$BUGCHECK_IGNORE_FLAGS and RDB_BUGCHECK_IGNORE_FLAGS

Flag	Description
G	Disables dumping of global buffer data structures. Disabling this information may make it more difficult to diagnose global buffer related bugchecks. For a database that has been opened with thousands of global buffers, this output can be quite large.
H	Disables dumping of root file information. This information is identical to the output from RMU Dump Header with the Debug option. Disabling this information substantially limits the possibility of diagnosing problems related to physical storage areas, recovery problems, and various I/O subsystem related bugchecks. This section is generally not very large. A database that has hundreds of storage areas may generate a significant amount of output in this section, but in general the quantity of dump output is not significant.
L	Disables dumping of locking information. This is a dump of the lock database for the system. Disabling the dumping of this information substantially limits the possibility of diagnosing bugchecks related to locking. On a system with many or large databases, this output can be significant.
P	Disables dumping of page buffers. All database pages in the attached user's buffers are normally dumped. Disabling this information can make it more difficult to diagnose I/O subsystem related errors. If the user has been allocated many hundreds or thousands of buffers, this output can be significant.

A.76 RDM$MAILBOX_CHANNEL

OpenVMS VAX OpenVMS Alpha

The logical name RDM$MAILBOX_CHANNEL contains the node-specific address of the database monitor mailbox; this address is used by processes to communicate with the appropriate database monitor.

The RDM$MAILBOX_CHANNEL logical name's value is established by the monitor process when the permanent mailbox is initially created; the value is not defined by the user.

The RDM$MAILBOX_CHANNEL logical name is translated using the LNM$PERMANENT_MAILBOX table or tables in the LNM$SYSTEM_TABLE logical name table.

Note

If the LNM$PERMANENT_MAILBOX table is not defined in the LNM$SYSTEM_TABLE logical name table, the following occurs:

- You receive the "monitor is not running" error when you try to attach to a database.
- The RMU Monitor Start command hangs.

By default, the LNM$PERMANENT_MAILBOX table is defined in the LNM$SYSTEM_TABLE logical name table. However, sometimes a user or third-party application redefines the LNM$PERMANENT_MAILBOX table in another logical name table (such as the LNM$GROUP table). To avoid these problems, perform the following steps:

1. Define the LNM$PERMANENT_MAILBOX table in the LNM$SYSTEM_TABLE:

   ```
   $ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$PERMANENT_MAILBOX LNM$SYSTEM
   ```

2. Start the database monitor:

   ```
   $ RMU/MONITOR START
   ```

3. Start the application

Alternately, you can change the application that redefines the LNM$PERMANENT_MAILBOX table so that LNM$PERMANENT_MAILBOX is defined as a search list that includes the LNM$SYSTEM_TABLE table, as shown in the following example:

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$PERMANENT_MAILBOX LNM$GROUP,-
_$ LNM$SYSTEM
```

♦

A.77 RDM$MONITOR and RDB_MONITOR

The logical name RDM$MONITOR and the configuration parameter RDB_MONITOR define the device and directory where the Oracle Rdb monitor log file is to reside; the value should not include a file name specification. The directory location defined by the value of the logical name or configuration parameter is tested and used only by the RMONSTART.COM command file.

Note

The logical name and configuration parameter are not actually used by the Oracle Rdb monitor; only the RMONSTART.COM command file uses the logical name and configuration parameter. The logical name and configuration parameter are primarily defined for use by layered products.

To manually change the monitor log file location, see the information on the RMU Monitor Start Output command in the *Oracle RMU Reference Manual*.

OpenVMS VAX OpenVMS Alpha

The RDB$MONITOR logical name is translated using the LNM$SYSTEM table. ♦

A.78 RDM$MON_USERNAME

OpenVMS VAX OpenVMS Alpha

The logical name RDM$MON_USERNAME designates the name of the user whose quotas the monitor process, upon startup, is to inherit.

During normal system startup, an RMU Monitor Start command is performed. If global buffers are enabled, the default PGFLQUOTA of 20,480 (the original process quotas) may not be sufficient for the monitor process to be created if many global buffers are used.

When a monitor is started using the RMU Monitor Start command, the quota limit that the monitor process uses is determined as the larger of three factors: a hard-coded “minimum-necessary” value, the quota value from the designated user, or the quota value from the user who is performing the startup.

For example, the hard-coded minimum value of the PGFLQUOTA process quota is 20,480. If the quota value for the user SYSTEM is 40,960 and the quota value for the user starting the monitor is 30,720, then the monitor is started with a PGFLQUOTA of 40,960.

The hard-coded minimum values for each monitor quota are as follows:

ASTLM	256
BIOLM	256
BYTLM	20,480
DIOLM	256
ENQLM	8,192
FILLM	1,024
PGFLQUOTA	20,480
PRCCNT	64
TQCNT	64
WSEXTENT	512
WSQUOTA	512

Example A–18 shows how to use the RDM$MON_USERNAME logical name to designate that the monitor process inherit the user quotas specified by the user account GOOD_QUOTAS, according to the algorithm previously mentioned.

Example A–18 Using the RDM$MON_USERNAME Logical Name

```
$ DEFINE RDM$MON_USERNAME GOOD_QUOTAS
```

The RDM$MON_USERNAME logical name was introduced because it is possible for an RMU Monitor Start operation at system startup to result in an exceeded quota error if global buffers are enabled. The RDM$MON_USERNAME logical name allows you to set up a special account with the appropriate quotas for executing the monitor process. ♦

A.79 RDMS$AUTO_READY and RDB_AUTO_READY

The RDMS$AUTO_READY logical name and the RDB_AUTO_READY configuration parameter allow a process requesting a logical area lock in CR (concurrent read) mode to obtain the lock in CU (concurrent update) mode if the process already holds a carry-over lock in CU mode for the logical area.

Under normal circumstances, a process may start an update transaction that reads a row first, and then updates it later. In this case, Oracle Rdb may first grant the process a logical area lock in CR (concurrent read) mode, then upgrade the lock to CU (concurrent update) mode later. If the process starts another update transaction, the same situation can occur. Many lock operations can be required to transition from CR to CU mode and back again.

You can enable update carry-over locks at the table level for a process by defining the RDMS$AUTO_READY logical name or the RDB_AUTO_READY configuration parameter for the process. If RDMS$AUTO_READY or RDB_AUTO_READY is defined, when the process requests a logical area lock in CR mode, it will obtain that lock in CU mode if it is already holding a carry-over lock on the logical area in CU mode. This optimization of carry-over locks for logical areas reduces the locking overhead for most update transactions.

Digital UNIX

Example A–19 shows how to enable update carry-over locking at a table level for a process.

Example A–19 Using the RDB_AUTO_READY Configuration Parameter

```
RDB_AUTO_READY 1
```

♦

Note that carry-over locking must be enabled when you define the RDMS$AUTO_READY logical name or the RDB_AUTO_READY configuration parameter to 1; otherwise, defining a value of 1 will not enable update carry-over locks at the table level.

A process benefits from update carry-over lock optimization for tables only if the process is updating the database. The benefit of enabling update carry-over locks at a table level for a process is that when the process starts a new update transaction on a table for which it holds a carry-over lock in CU mode for the logical area, it will get the logical area lock in CU mode. Getting the logical area lock in CU mode at the start of the transaction means the process avoids the locking overhead of first downgrading and then upgrading the lock later. If the process does not perform any updates, getting the logical area locks in the lower (CR) mode is sufficient.

The RDMS$AUTO_READY logical name and the RDB_AUTO_READY configuration parameter should be used in high volume, update-intensive, transaction processing environments when the Performance Monitor indicates that a large number of lock conversions are taking place for each transaction.

A process that has enabled update carry-over locking at the table level can cause concurrency problems if the process reserves tables in PROTECTED READ or PROTECTED WRITE modes, or if it performs sequential scans of tables.

OpenVMS VAX OpenVMS Alpha

Example A–20 shows how to disable update carry-over locking at a table level for a process by deassigning the RDMS$AUTO_READY logical name.

Example A–20 Disabling Update Carry-Over Locking at a Table Level by Deassigning the RDMS$AUTO_READY Logical Name

```
$ DEASSIGN RDMS$AUTO_READY
```

♦

A.80 RDMS$BIND_OUTLINE_FLAGS and RDB_BIND_OUTLINE_FLAGS

The logical name RDMS$BIND_OUTLINE_FLAGS and the configuration parameter RDB_BIND_OUTLINE_FLAGS cause Oracle Rdb to ignore query outlines. If you want the optimizer to ignore any outlines that may be stored for a query, define RDMS$BIND_OUTLINE_FLAGS or RDB_BIND_OUTLINE_FLAGS to the value “I”. When a process has defined the logical name or configuration parameter as “I”, the optimizer will ignore any stored outlines when processing the query.

Section 5.9 provides more information on query outlines.

A.81 RDMS$BIND_OUTLINE_MODE and RDB_BIND_OUTLINE_MODE

When multiple outlines exist for a query, you set the logical name RDMS$BIND_OUTLINE_MODE or the configuration parameter RDB_BIND_OUTLINE_MODE to the value of the outline mode for the outline you want the optimizer to use.

Suppose, for example, two outlines are stored for a particular query. Assume that one outline has the default outline mode value of 0 and the other outline has an outline mode value of –1. If you want the optimizer to use the outline with the outline mode value of 0 for the query, the RDMS$BIND_OUTLINE_MODE logical name or the RDB_BIND_OUTLINE_MODE configuration parameter should be set to 0 (zero). If you want the optimizer to use the other outline for the query, the RDMS$BIND_OUTLINE_MODE logical name or the RDB_BIND_OUTLINE_MODE configuration parameter should be set to –1.

Section 5.9 provides more information on query outlines.

A.82 RDMS$BIND_PRESTART_TXN and RDB_BIND_PRESTART_TXN

The RDMS$BIND_PRESTART_TXN logical name and the RDB_BIND_PRESTART_TXN configuration parameter allow you to establish the default setting for prestarted transactions outside of an application.

The logical name and configuration parameter allow prestarted transactions to be disabled in environments that do not allow source code to be modified, or in which the source code may not be available.

The value 1 indicates that prestarted transactions are enabled, and the value 0 indicates that prestarted transactions are disabled.

OpenVMS VAX OpenVMS Alpha

The following example disables prestarted transactions on OpenVMS.

```
$ DEFINE RDMS$BIND_PRESTART_TXN 0
```

♦

You can override the value specified for RDMS$BIND_PRESTART_TXN or RDB_BIND_PRESTART_TXN by using the PRESTARTED TRANSACTION clause in your application.

Note

Oracle Rdb recommends that most applications accept the default setting of PRESTARTED TRANSACTIONS ARE ON, which provides reduced I/O during processing of the SET TRANSACTION statement.

A.83 RDMS$BIND_QG_CPU_TIMEOUT and RDB_BIND_QG_CPU_TIMEOUT

The RDMS$BIND_QG_CPU_TIMEOUT logical name and the RDB_BIND_QG_CPU_TIMEOUT configuration parameter restrict the amount of CPU time used to optimize a query for execution. If the query is not optimized and prepared for execution before the CPU time limit is reached, an error message is returned and the query is aborted.

The default is unlimited CPU time for query compilation. Dynamic SQL options are inherited from the compilation qualifier.

Digital UNIX

Include the line shown in Example A–21 in your configuration file to limit the elapsed CPU time spent by the optimizer to 5 seconds.

Example A–21 Using the RDB_BIND_QG_CPU_TIMEOUT Configuration Parameter

```
RDB_BIND_QG_CPU_TIMEOUT 5
```

♦

This logical name and configuration parameter are translated at attach time and supersede all options specified in an application.

A.84 RDMS$BIND_QG_REC_LIMIT and RDB_BIND_QG_REC_LIMIT

You can use the RDMS$BIND_QG_REC_LIMIT logical name or the RDB_BIND_QG_REC_LIMIT configuration parameter to establish a process or system limit on the number of rows a query returns. If a user enters a query and the returned rows exceed the limit set by RDMS$BIND_QG_REC_LIMIT or RDB_BIND_QG_REC_LIMIT, the user receives an error message and the query is aborted. This prevents users from overloading the system with general queries that return every row in a table or every row in a multiple table join. You can also set a time limit to accomplish this goal; refer to Section A.85.

OpenVMS VAX ≡ OpenVMS Alpha ≡

Example A–22 shows how to set a returned row limit.

Example A–22 Using the RDMS$BIND_QG_REC_LIMIT Logical Name

```
$ DEFINE RDMS$BIND_QG_REC_LIMIT 1000
```

♦

Example A–22 limits the number of returned rows to 1000. The specified value is independent of the number of table rows read by the query, for example intermediate rows read during a join operation; it applies only to the number of rows returned. However, you should avoid setting too low a value because the row limit value applies to all database queries, including queries to system tables (by SQL, for example) that fetch the metadata required to parse a query. If the row limit value prevents SQL from fetching all the required metadata, the attempt to execute the query fails.

RDMS$BIND_QG_REC_LIMIT and RDB_BIND_QG_REC_LIMIT are translated when a process attaches to the database, and their value supersedes any limit specified within an application using compiler qualifiers.

A.85 RDMS$BIND_QG_TIMEOUT and RDB_BIND_QG_TIMEOUT

You can use the RDMS$BIND_QG_TIMEOUT logical name or the RDB_BIND_QG_TIMEOUT configuration parameter to establish a system limit on the amount of time the optimizer spends compiling a query. If a user enters a query and the elapsed time specified by RDMS$BIND_QG_TIMEOUT or RDB_BIND_QG_TIMEOUT is exceeded, the user receives an error message and the query is aborted. This prevents users from overloading the system with general queries that return every row in a table or every row in a multiple table join. You can also establish a limit on the number of returned rows; refer to Section A.84.

Digital UNIX

Example A–23 shows how to set a system limit on the amount of time the optimizer spends compiling a query.

Example A–23 Using the RDB_BIND_QG_TIMEOUT Configuration Parameter

```
RDB_BIND_QG_TIMEOUT 15
```

♦

Example A–23 limits the elapsed time spent by the optimizer to 15 seconds.

RDMS$BIND_QG_TIMEOUT and RDB_BIND_QG_TIMEOUT are translated when a process attaches to the database, and their value supersedes any limit specified within an application using compiler qualifiers.

A.86 RDMS$BIND_SEGMENTED_STRING_BUFFER and RDB_BIND_SEGMENTED_STRING_BUFFER

The RDMS$BIND_SEGMENTED_STRING_BUFFER logical name and the RDB_BIND_SEGMENTED_STRING_BUFFER configuration parameter allow you to reduce the overhead of I/O operations when you manipulate segmented strings.

You may be able to increase the efficiency of applications that manipulate segmented strings by increasing the buffer space for segmented strings.

OpenVMS VAX OpenVMS Alpha

When you use the RDML and RDBPRE precompilers, be sure to define a sufficiently large value for the RDMS$BIND_SEGMENTED_STRING_BUFFER logical name. ♦

You need an adequate buffer size to store large segmented strings (using segmented string storage maps) in storage areas other than the default RDB$SYSTEM storage area. The minimum acceptable value for the RDMS$BIND_SEGMENTED_STRING_BUFFER logical name and the RDB_BIND_SEGMENTED_STRING_BUFFER configuration parameter must be equal to the sum of the length of the segments of the segmented string. For example, if you know that the sum of the length of the segments is 1 megabyte, then 1,048,576 bytes is an acceptable value for this logical name or configuration parameter.

OpenVMS VAX OpenVMS Alpha

You must specify the logical name value because when RDML and RDBPRE precompilers store segmented strings, Oracle Rdb does not know which table contains the string until after the entire string is stored. Oracle Rdb buffers the entire segmented string, if possible, and does not store it until the STORE statement executes. ♦

If the segmented string remains buffered, it is stored in the appropriate storage area. If the string is not buffered (because it is larger than the defined value for the logical name, configuration parameter, or the default value of 10,000 bytes), it is not stored in the default storage area and the following exception message is displayed:

```
%RDB-F-IMP_EXC, facility-specific limit exceeded
-RDMS-E-SEGSTR_AREA_INC, segmented string was stored incorrectly
```

To avoid this error, set the value of the RDMS$BIND_SEGMENTED_STRING_BUFFER logical name or the RDB_BIND_SEGMENTED_STRING_BUFFER configuration parameter to a sufficiently large value. Note that a value of up to 500 megabytes can be specified for this logical name and configuration parameter.

> **Note**
>
> The SQL interface for lists (segmented strings) does not require you to define the value for this logical name or configuration parameter. Before the list is brought into the buffer, SQL knows the column that the list is associated with and the table in which it is stored. However, for large lists, defining this logical name or configuration parameter with a value large enough to hold the entire list may improve the handling performance of storing the list.

OpenVMS VAX OpenVMS Alpha

Example A–24 shows how to change this value to 20000 bytes.

Example A–24 Using the RDMS$BIND_SEGMENTED_STRING_BUFFER Logical Name

```
$ DEFINE RDMS$BIND_SEGMENTED_STRING_BUFFER 20000
```

♦

A.87 RDMS$BIND_SEGMENTED_STRING_COUNT and RDB_BIND_SEGMENTED_STRING_COUNT

The RDMS$BIND_SEGMENTED_STRING_COUNT logical name and the RDB_BIND_SEGMENTED_STRING_COUNT configuration parameter specify the allocation size, expressed as the number of entries, in the segmented string ID list; the ID list is used to materialize and manipulate segmented strings for a table row.

By default, the segmented string ID list is allocated in multiples of 64 entries, with each entry addressing a separate segmented string. If a table contains more than the indicated number of segmented strings, another segmented string ID list whose size is twice the previous size will be allocated; the first block will contain 64 entries, the second block will contain 128 entries, and so forth. To avoid fragmenting virtual memory, the logical name or configuration parameter value should closely approximate the number of segmented strings typically retrieved by the application.

Define RDMS$BIND_SEGMENTED_STRING_COUNT or RDB_BIND_SEGMENTED_STRING_COUNT as *n*, where *n* is the number of segmented strings in the table. The default and minimum value is 64.

Digital UNIX

Example A–25 defines a value of 100 for this configuration parameter.

Example A–25 Using the RDB_BIND_SEGMENTED_STRING_COUNT Configuration Parameter

```
RDB_BIND_SEGMENTED_STRING_COUNT 100
```

♦

Defining this logical name or configuration parameter can help to avoid a problem in which an import operation fails and the process loops, producing these messages:

```
%SQL-F-BADBLOB, unable to import a segmented string
%RDB-F-SYS_REQUEST, error from system services request
-RDMS-F-EXQUOTA, exceeded quota
-SYSTEM-F-EXQUOTA, exceeded quota
```

A problem has been found when this logical name or configuration parameter is used with multiple database attaches. Each additional attach causes virtual memory to be allocated unnecessarily. This extra virtual memory is not released until image rundown.

Oracle Corporation suggests that this logical name and configuration parameter not be used if more than one database attach is performed per session. Alternatively, because this logical name and configuration parameter are usually associated with importing large databases with many segmented strings, make sure you exit the SQL interactive utility between IMPORT and ATTACH statements.

A.88 RDMS$BIND_SEGMENTED_STRING_DBKEY_SCOPE and RDB_BIND_SEGMENTED_STRING_DBKEY_SCOPE

The RDMS$BIND_SEGMENTED_STRING_DBKEY_SCOPE logical name and the RDB_BIND_SEGMENTED_STRING_DBKEY_SCOPE configuration parameter indicate whether the dbkey of a modified segmented string may be reused by the process.

The dbkey of the first segment of a segmented string is stored in the user's data record. It is possible that certain applications test this dbkey value to determine if something has changed; if Oracle Rdb were to reuse the same segment, then this could break the user application.

When the RDMS$BIND_SEGMENTED_STRING_DBKEY_SCOPE logical name or the RDB_BIND_SEGMENTED_STRING_DBKEY_SCOPE configuration parameter is set to 1 (indicating true), the dbkeys of modified segmented strings will not be reused by the process; the dbkeys will be available for reuse after the process detaches from the database. When the logical name or configuration parameter is undefined, or set to 0 (indicating false), the dbkeys of modified segmented strings are reused by the process. The default value is 0 (false).

Note

If the database has the Replication Option for Rdb (formally known as Data Distributor) transfers enabled, the logical name RDMS$BIND_SEGMENTED_STRING_DBKEY_SCOPE or the configuration parameter RDB_BIND_SEGMENTED_STRING_DBKEY_SCOPE value is always 1 (true).

OpenVMS VAX OpenVMS Alpha

Example A–26 shows how to change this value to indicate that segmented string dbkeys are not reused until the process detaches from the database.

Example A–26 Using the RDMS$BIND_SEGMENTED_STRING_DBKEY_SCOPE Logical Name

```
$ DEFINE RDMS$BIND_SEGMENTED_STRING_DBKEY_SCOPE 1
```

♦

A.89 RDMS$BIND_SORT_WORKFILES and RDB_BIND_SORT_WORKFILES

OpenVMS VAX OpenVMS Alpha

The logical name RDMS$BIND_SORT_WORKFILES specifies how many work files the OpenVMS Sort utility (SORT) is to use if work files are required. The SORT default is 2 and the maximum number is 10. The work files can be individually controlled by the SORTWORK*n* logical names (where *n* is from 0 through 9).

You can increase the efficiency of Oracle Rdb sort operations, which use the SORT utility, by assigning the location of the temporary sort work files to different disks. These assignments are made by using up to ten logical names, SORTWORK0 through SORTWORK9.

Normally, SORT places work files in the user's SYS$SCRATCH directory. By default, SYS$SCRATCH is the same device and directory as the SYS$LOGIN location. When many concurrent users enter Oracle Rdb queries that necessitate sort operations, and those users share the same disk, performance can suffer if disk I/O operation is a bottleneck. Specifying that a user's work files will reside on separate disks permits overlap of the SORT read/write cycle. Also, you may encounter cases where insufficient space exists on the SYS$SCRATCH disk device (for example, while Oracle Rdb builds indexes for a very large table); using the SORTWORK0 through SORTWORK9 logical names can help you avoid this problem.

For the greatest sorting efficiency, you can place work files on:

- The fastest device available
- The device having the least activity
- The device having the most space available
- Separate devices

At DCL level, you can select a different device (as shown in Example A–27) for any work file by specifying the device.

Example A–27 Using the SORTWORKn Logical Name

```
$ DEFINE SORTWORKn   device:
```

The logical name SORTWORK*n* is a work file, where *n* indicates the number of the work file. By default, SORT creates just two work files. The RDMS$BIND_SORT_WORKFILES logical name uses this default, unless more work files are specified. Thus, for Oracle Rdb users, you can assign up to ten (SORTWORK0 through SORTWORK9) logical names. You can use the logical name SORTWORK*n* in two ways:

- If you define SORTWORK*n* and specify a device, Oracle Rdb creates a hidden temporary file. No root directory is necessary, but users must have disk quotas enabled on the specified device.
- If you define SORTWORK*n* and specify a device and a directory name, Oracle Rdb creates a visible temporary file. Users do not need disk quotas enabled on the specified device. Instead, you can add an ACL for a resource identifier and grant the resource identifier to each user. The disk quota for the resource identifier is then used. This is a good alternative when a large number of database users may be concurrently using the sort work files, because this alternative requires only one directory for all of the active users.

Example A–28 assigns the first work file to the $1$DUA1 device.

Example A–28 Using the SORTWORK0 Logical Name

```
$ DEFINE SORTWORK0   $1$DUA1:
```

If the user's writable directory exists, the $1$DUA1:[USER1]SORTWORK.TMP file is created. If no writable directory exists on the target disk, the operation fails.

Consider the following example:

- The OpenVMS files that comprise an Oracle Rdb database called CORPORATE_SALARY are distributed over six RA81 disk devices: $222$DUA12, $222$DUA14, $222$DUA16, $222$DUA18, $222$DUA21, and $222$DUA22. At daily peak usage times, and particularly when the weekly payroll is being computed, activity on these disks is very high.

- The SYS$SCRATCH logical names for most users accessing this database translate into directories on several I/O-saturated disks: $222$DUA17, $222$DUA19, and $222$DUA20. In addition, because of other uses for these disks, unused disk space is not always available.
- After you analyze disk I/O activity with the OpenVMS Monitor utility on all your available disks, you determine that $222$DUA8, $222$DUA9, $222$DUA10, and $222$DUA11 have the least I/O activity. These disks contain no OpenVMS files from the CORPORATE_SALARY database and are not used as SYS$SCRATCH locations by any users (or users running programs) who access this database.
- Because of the nature of this application, there is no easy way to avoid the high volume of concurrent SORT operations.

Entering the four DCL DEFINE commands shown in Example A–29 causes the SORT work files to be placed on separate disks from the user's SYS$LOGIN location, which, as previously described, may improve sort performance.

Example A–29 Using SORTWORK*n* to Specify Multiple Devices

```
$ DEFINE SORTWORK0 $222$DUA8:
$ DEFINE SORTWORK1 $222$DUA9:
$ DEFINE SORTWORK2 $222$DUA10:
$ DEFINE SORTWORK3 $222$DUA11:
```

♦

Digital UNIX

To increase the efficiency of Oracle Rdb sort operations that use the Sort utility, you can spread work files across disks by assigning the location of the temporary sort work files to different disks. You make these assignments by using the environmental variables SORTWORK0 through SORTWORK255. For example, you can define the environmental variables by using the following commands:

```
setenv  SORTWORK0 /tmp
setenv  SORTWORK1 .
```

To specify how many work files the Sort utility is to use, define the RDB_BIND_SORT_WORKFILES configuration parameter in the .dbsrc configuration file. The following example shows how to specify that the Sort utility uses 9 files:

```
RDB_BIND_SORT_WORKFILES 9
```

♦

A.90 RDMS$BIND_VALIDATE_CHANGE_FIELD and RDB_BIND_VALIDATE_CHANGE_FIELD

When you use the SQL ALTER DOMAIN statement, Oracle Rdb changes the metadata in the system relation but converts the data stored in the table when the next update operation occurs. This approach usually causes no problems. Sometimes, however, you may make changes to metadata that render data unreadable due to conversion problems between stored data and newly defined metadata.

By defining the RDMS$BIND_VALIDATE_CHANGE_FIELD logical name or the RDB_BIND_VALIDATE_CHANGE_FIELD configuration parameter, you can ensure that data records are validated and converted to the new metadata definition by the ALTER DOMAIN statement, rather than at data update time.

Digital UNIX

Example A–30 shows how to define the RDB_BIND_VALIDATE_CHANGE_FIELD configuration parameter in the configuration file.

Example A–30 Using the RDB_BIND_VALIDATE_CHANGE_FIELD Configuration Parameter

```
RDB_BIND_VALIDATE_CHANGE_FIELD 1
```

♦

A.91 RDMS$BIND_WORK_FILE and RDB_BIND_WORK_FILE

You can use the RDMS$BIND_WORK_FILE logical name or the RDB_BIND_WORK_FILE configuration parameter to redirect the location of temporary files that Oracle Rdb sometimes creates for use in matching operations. You have the following three options:

OpenVMS VAX OpenVMS Alpha

- If you do not define RDMS$BIND_WORK_FILE, Oracle Rdb creates the temporary file on the device specified in SYS$LOGIN. ♦
- If you define RDMS$BIND_WORK_FILE or RDB_BIND_WORK_FILE and specify a device name, Oracle Rdb creates the temporary file in your default directory on the specified device.

OpenVMS VAX OpenVMS Alpha

On OpenVMS, this reduces the number of disk I/O operations in SYS$LOGIN. ♦

- If you define RDMS$BIND_WORK_FILE or RDB_BIND_WORK_FILE and specify a device name and directory, Oracle Rdb creates the temporary file in the specified directory. This enables the file to use resource identifiers for disk quota allocation.

In all three cases, Oracle Rdb creates the file when the process running the query runs out of virtual memory, and deletes the file once the query is completed.

OpenVMS VAX OpenVMS Alpha

Example A–31 shows how to assign the location of temporary files to a specific device and directory.

Example A–31 Using the RDMS$BIND_WORK_FILE Logical Name

```
$ DEFINE RDMS$BIND_WORK_FILE WORK$DISK:[RDB.WORK]
```

♦

If you assign temporary files to a device and directory and your system fails before Oracle Rdb can delete a temporary file, you can find the file and delete it yourself.

OpenVMS VAX OpenVMS Alpha

The following example uses the DCL DIRECTORY command to find the file specified in Example A–31:

```
$ DIRECTORY WORK$DISK:[RDB.WORK]

Directory WORK$DISK:[RDB.WORK]

RDMSTTBL$WYQD02QQU4D.TMP;1     76 30-JUN-1991 19:47:04.20

Total of 1 file, 76 blocks.
```

The temporary file name consists of RDMSTTBL$ followed by a random sequence of characters, and a file type of .tmp.

Because the Oracle Rdb work file is an OpenVMS RMS file, you can also set the RMS multibuffer and multiblock counts. Use the DCL SET RMS_DEFAULT command to improve the performance of Oracle Rdb temporary files by specifying appropriate values for the /BUFFER_COUNT and /BLOCK_COUNT qualifiers. ♦

The RDMS$BIND_WORK_FILE logical name and the RDB_BIND_WORK_FILE configuration parameter are often used in conjunction with the RDMS$BIND_WORK_VM logical name and the RDB_BIND_WORK_VM configuration parameter.

A.92 RDMS$BIND_WORK_VM and RDB_BIND_WORK_VM

The RDMS$BIND_WORK_VM logical name and the RDB_BIND_WORK_VM configuration parameter permit you to reduce the overhead of disk I/O for matching operations by letting you specify the amount of virtual memory (VM), in bytes, to be allocated to your process. Once the allocation is exhausted, additional data values will be written to a temporary file on disk.

OpenVMS VAX OpenVMS Alpha

If RDMS$BIND_WORK_FILE is undefined, the temporary file is located in SYS$LOGIN. ♦

The default is 10,000 bytes. The maximum allowed value is restricted only by the amount of memory available on your system.

Digital UNIX

Example A–32 defines the RDB_BIND_WORK_VM value to be 25,000 bytes.

Example A–32 Using the RDB_BIND_WORK_VM Configuration Parameter

```
RDB_BIND_WORK_VM 25000
```

♦

See Section 3.2.1.7 and Section 8.1.3 for more information.

A.93 RDMS$DEBUG_FLAGS and RDB_DEBUG_FLAGS

The RDMS$DEBUG_FLAGS logical name and the RDB_DEBUG_FLAGS configuration parameter allow you to examine database access strategies and the estimated cost of those strategies when your program is run. See Section 5.8.7 and Appendix C for more information.

A.94 RDMS$DEBUG_FLAGS_OUTPUT and RDB_DEBUG_FLAGS_OUTPUT

The RDMS$DEBUG_FLAGS_OUTPUT logical name and the RDB_DEBUG_FLAGS_OUTPUT configuration parameter allow you to name an output file in which to collect the output from RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS when you run your program. You must have write access to the directory and the disk device must exist for this logical name or configuration parameter to be successful.

A.95 RDMS$DIAG_FLAGS and RDB_DIAG_FLAGS

You can use the RDMS$DIAG_FLAGS logical name or the RDB_DIAG_FLAGS configuration parameter to assist in locating erroneous queries. When RDMS$DIAG_FLAGS or RDB_DIAG_FLAGS is defined, the query compiler checks for sort keys totally defined by contexts external to the record selection expression (RSE) that includes the sort clause. When such a case is encountered, the query compiler produces the following error:

```
%RDB-E-INVALID_BLR, request BLR is incorrect at offset n
-RDMS-F-SORTKEYEXT, sort key is external to RSE context
```

Digital UNIX

Include the line shown in Example A–33 in your configuration file to assist in locating erroneous queries.

Example A–33 Using the RDB_DIAG_FLAGS Configuration Parameter

```
RDB_DIAG_FLAGS S
```

♦

Oracle Rdb allows you to mix old format (chained) and new format (indexed) segmented strings in the same table. You may want to convert your chained segmented strings to the new indexed format. Conversion is desirable if, for example, you want to perform FETCH LAST statements with a scrolling list cursor. With chained segmented strings, a FETCH LAST statement would cause Oracle Rdb to read all segments before reaching the desired segment, which is not optimal. With indexed segmented strings, a FETCH LAST statement would cause Oracle Rdb to read only the pointer segment and the last data segment.

OpenVMS VAX OpenVMS Alpha

Define RDMS$DIAG_FLAGS or RDB_DIAG_FLAGS as L to prevent Oracle Rdb from performing a FETCH LAST statement with chained segmented strings.

Example A–34 Using the RDMS$DIAG_FLAGS Logical Name

```
$ DEFINE RDMS$DIAG_FLAGS L
```

♦

Defining RDMS$DIAG_FLAGS or RDB_DIAG_FLAGS to L causes an OPEN CURSOR statement to fail if it tries to open a SCROLL list cursor.

A.96 RDMS$KEEP_PREP_FILES

OpenVMS VAX OpenVMS Alpha

You can use the RDMS$KEEP_PREP_FILES logical name to cause the RDBPRE preprocessor to retain the intermediate .mar and language files. This can be helpful when you are trying to debug an RDBPRE program and need to refer to the language files. Use the command shown in Example A–35 to specify that you want to retain these files.

Example A–35 Using the RDMS$KEEP_PREP_FILES Logical Name

```
$ DEFINE RDMS$KEEP_PREP_FILES YES
```

♦

A.97 RDMS$RUJ and RDB_RUJ

You can use the RDMS$RUJ logical name or the RDB_RUJ configuration parameter to locate the .ruj file on a different disk and directory from the default directory. This can help to reduce contention in that directory.

Digital UNIX

Include the RDB_RUJ configuration parameter in the configuration file to specify the location you want to use, as shown in Example A–36.

Example A–36 Using the RDB_RUJ Configuration Parameter

```
RDB_RUJ /usr/clients/journal
```

♦

See Section 3.2.1.7 and Section 8.1.2 for more information.

A.98 RDMS$USE_OLD_CONCURRENCY and RDB_USE_OLD_CONCURRENCY

Prior to V4.2, the SQL interface provided syntax to select different isolation levels but the syntax was automatically upgraded to isolation level serializable (degree 2 consistency). Many 4GLs also selected this mode by default.

In Oracle Rdb V4.2, full isolation level support was added, and these low isolation levels were supported.

When applications using these low isolation levels were upgraded to V4.2, they encountered significant changes—more locks were used and different records became visible. Although this behavior is expected for CONSISTENCY LEVEL 2 applications, the applications used prior to V4.2 were never tested or tuned under such an environment.

A solution was implemented with Oracle Rdb V4.2A and V5.1 to allow applications to revert to V4.1 behavior by defining RDMS$USE_OLD_CONCURRENCY to 1. This converts TPB$K_DEGREE2 to TPB$K_DEGREE3 to maintain the V4.1 behavior. Note that TPB$K_ISOLATION_1 (ISOLATION LEVEL READ COMMITTED) is not affected by this logical name and configuration parameter.

Oracle Corporation recommends that applications and 4GLs be adjusted to use the isolation level best suited for their environment as soon as practical.

OpenVMS VAX ≡ OpenVMS Alpha ≡

In Example A–37, the RDMS$USE_OLD_CONCURRENCY logical name causes Oracle Rdb to use the V4.1 isolation level behavior. Defining the RDMS$DEBUG_FLAGS logical name to T causes the isolation levels selected by Oracle Rdb to be displayed.

Example A–37 Using the RDMS$USE_OLD_CONCURRENCY Logical Name to Cause Oracle Rdb to Use V4.1 Isolation Level Behavior

```
$ DEFINE RDMS$DEBUG_FLAGS "T"
$ DEFINE RDMS$USE_OLD_CONCURRENCY 1
$ SQL
SQL> ATTACH 'FILE mf_personnel';
SQL> SET TRANSACTION READ WRITE CONSISTENCY LEVEL 2;
%SQL-I-DEPR_FEATURE, Deprecated Feature: CONCURRENCY or CONSISTENCY LEVEL.  Use
ISOLATION LEVEL instead
 Compile transaction on db: X00000001
~T Transaction Parameter Block: (len=3)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_DEGREE2 (read committed)
0002 (00002) TPB$K_WRITE (read write)
~T Concurrency option (TPB$K_DEGREE2) converted to TPB$K_DEGREE3
 Start_transaction on db: X00000001, db count=1
SQL> EXIT
 Commit_transaction on db: X00000001
 Prepare_transaction on db: X00000001
```

Oracle Rdb has always used the read/write transaction mode when snapshots are disabled for a database. Oracle Rdb V6.0 or higher outputs an indication of this when the T debug flag is in use, as shown in Example A–38.

Example A–38 Using the RDMS$USE_OLD_CONCURRENCY and RDMS$DEBUG_FLAGS Logical Names to Display the Conversion of Read-Only Transactions When Snapshots Are Disabled

```
$ DEFINE RDMS$DEBUG_FLAGS "T"
$ DEFINE RDMS$USE_OLD_CONCURRENCY 1
$ SQL
SQL> ALTER DATABASE FILE mf_personnel
cont> SNAPSHOT IS DISABLED;
SQL> ATTACH 'FILE mf_personnel';
SQL> SET TRANSACTION READ ONLY;
 Compile transaction on db: X00000002
~T Transaction Parameter Block: (len=2)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_READ (read only)
 Start_transaction on db: X00000002, db count=1
~T Snapshots are disabled, READ ONLY converted to READ WRITE
SQL>
```

♦

A.99 RDMS$USE_OLD_COST_MODEL and RDB_USE_OLD_COST_MODEL

When you define RDMS$USE_OLD_COST_MODEL or RDB_USE_OLD_COST_MODEL to be any value, the optimizer will not use workload or storage statistics. Using RDMS$USE_OLD_COST_MODEL or RDB_USE_OLD_COST_MODEL allows you to:

- Test the optimization of a query workload with and without the use of storage and workload statistics
- Selectively disable the use of workload and storage statistics for particular users, processes, and batch jobs

When the RDMS$USE_OLD_COST_MODEL logical name or the RDB_USE_OLD_COST_MODEL configuration parameter is defined, the optimizer uses cost and cardinality functions that were in use prior to V7.0 and ignores any workload and storage statistics that have been collected.

Deassign the logical name or configuration parameter to enable the optimizer to start using workload and storage statistics again in cost and cardinality estimation.

A.100 RDMS$USE_OLD_COUNT_RELATION and RDB_USE_OLD_COUNT_RELATION

In previous versions of Oracle Rdb, a CREATE INDEX statement would check whether a table was empty by fetching the first row. If the table was empty, Oracle Rdb could avoid collection and sorting of the data and creation of the index. This optimization works very well in uniform storage areas where the SPAM pages allow fast access to the first row.

However, in mixed-format storage areas, each page must be read and checked for an occurrence of a row for the table. In particular, when the table is partitioned across many areas, CREATE INDEX could execute many I/Os to determine whether or not a table was empty.

Oracle Rdb V7.0 includes an optimization within CREATE INDEX to avoid this area scan for all areas. If the table was created in the current transaction, sufficient internal information exists for Oracle Rdb to know if a table does not contain data. Oracle Rdb maintains an internal list of newly created tables to support this optimization.

If you attach to the database using RESTRICTED ACCESS, Oracle Rdb carries this optimization through until you disconnect from the database. RESTRICTED ACCESS is necessary so that Oracle Rdb can guarantee that no other process has updated the table after it was created during the current session. Because the IMPORT statement, by default, attaches to the new database for RESTRICTED ACCESS, this optimization helps improve the import of large databases. In particular, less I/O is needed to import a table that is placed using a hashed index if the table is mapped to mixed-format areas.

In some cases, where applications create and drop many tables, the maintenance of the internal list of new tables may not be desirable. In those cases, you can define the logical name RDMS$USE_OLD_COUNT_RELATION or the configuration parameter RDB_USE_OLD_COUNT_RELATION to disable this optimization. Only the existence of the logical name or configuration parameter is required; it can be defined as any value.

A.101 RDMS$USE_OLD_SEGMENTED_STRING and RDB_USE_OLD_SEGMENTED_STRING

Defining the RDMS$USE_OLD_SEGMENTED_STRING logical name or the RDB_USE_OLD_SEGMENTED_STRING configuration parameter retains the old format (chained) segmented strings as the default. This logical name and configuration parameter cause the application to write the chained format segmented strings to all read/write media. Note that if a write-once storage area is used, then the new (indexed) segmented string format is always used. The value for the logical name or configuration parameter is not examined by Oracle Rdb.

OpenVMS VAX ≡ OpenVMS Alpha ≡

To enable this logical name, define it as shown in Example A–39.

Example A–39 Enabling the RDMS$USE_OLD_SEGMENTED_STRING Logical Name

```
$ DEFINE RDMS$USE_OLD_SEGMENTED_STRING YES
```

If later you wish to use the new (indexed) segmented string format, then you must deassign this logical name as shown in Example A–40.

Example A–40 Disabling the RDMS$USE_OLD_SEGMENTED_STRING Logical Name

```
$ DEASSIGN RDMS$USE_OLD_SEGMENTED_STRING
```

♦

Mixing of old and new segmented strings format is supported.

A.102 RDMS$USE_OLD_UPDATE_RULES and RDB_USE_OLD_UPDATE_RULES

Since V4.1, Oracle Rdb has used new update rules, which are enforced by default. With these new update rules it is not possible to modify or delete rows from a table that are directly joined with other tables. However, rows from a table can still be modified or deleted if the table is joined with other tables that are in a subquery. Because SQL does not have syntax that allows rows from a table to be modified or deleted and at the same time allows that table to be joined with other tables, the new update rules introduced with V4.1 have no affect on SQL applications.

OpenVMS VAX ≡ OpenVMS Alpha ≡

However, RDO applications that contain join update queries—that is, the update queries that modify or delete rows from a table that is joined with other tables—are affected by the new update rules and should be fixed.

Oracle Rdb now gives an error diagnostic for the following join update query:

```
FOR E IN EMPLOYEES CROSS D IN DEGREES OVER EMPLOYEE_ID
     WITH D.DEGREE = 'MA'
   ERASE E
END_FOR
%RDMS-E-JOIN_CTX_UPD, relation EMPLOYEES is part of a join, cannot be updated
```

In this previous update query, if an employee has two MA degrees, the same employee row will be joined to two different degree rows. Therefore, Oracle Rdb will try to delete the same row twice. Or, if instead of using the ERASE verb, the previous update query used a MODIFY verb on the EMPLOYEES table, then Oracle Rdb might modify the row more than once.

Prior to V4.1, join update queries similar to the previous query worked correctly or produced an error diagnostic trying to delete the same row more than once, or—even worse—produced a bugcheck. Some 4GLs and third-party software products were also affected.

The previous update query can be reworded into an equivalent form as follows:

```
FOR E IN EMPLOYEES WITH (ANY D IN DEGREES WITH D.EMPLOYEE_ID = E.EMPLOYEE_ID
                             AND D.DEGREE = 'MA')
   ERASE E
END_FOR
```

The rows can now be erased because the EMPLOYEES table is no longer directly joined to the DEGREES table. The use of a modified update query guarantees that an employee row will not be deleted more than once.

Oracle Rdb V4.1 and later versions support either new or old update rules. By default, V4.1 and higher versions enforce new update rules. To make Oracle Rdb continue to use the old update rules, define the logical name RDMS$USE_OLD_UPDATE_RULES to be 1.

Example A–41 Using the RDMS$USE_OLD_UPDATE_RULES Logical Name

```
$ DEFINE RDMS$USE_OLD_UPDATE_RULES 1
```

The following join update query will no longer work with the new update rules. Also, this update query will modify some salary history rows more than once and gives multiple salary raises to some managers!

```
! Give a 10% salary raise to all managers who have an MA degree.
FOR S IN SALARY_HISTORY CROSS D IN DEGREES CROSS DP IN DEPARTMENTS
   WITH S.EMPLOYEE_ID = D.EMPLOYEE_ID AND
        S.EMPLOYEE_ID = DP.MANAGER_ID AND
        S.SALARY_END MISSING          AND
        D.DEGREE = 'MA'
   MODIFY S USING S.SALARY_AMOUNT = S.SALARY_AMOUNT * 1.1
END_FOR
```

This query can be reworded using a subquery as follows:

```
FOR S IN SALARY_HISTORY
   WITH S.SALARY_END MISSING AND
        (ANY D IN DEGREES CROSS DP IN DEPARTMENTS
         WITH S.EMPLOYEE_ID = D.EMPLOYEE_ID AND
         S.EMPLOYEE_ID = DP.MANAGER_ID AND
         D.DEGREE = 'MA')
   MODIFY S USING S.SALARY_AMOUNT = S.SALARY_AMOUNT * 1.1
END_FOR
```

This revised query will work with the new as well as the old update rules, and it will ensure that each qualified manager gets a single salary raise. ♦

A.103 RDMS$VALIDATE_ROUTINE and RDB_VALIDATE_ROUTINE

Use the RDMS$VALIDATE_ROUTINE logical name or the RDB_VALIDATE_ROUTINE configuration parameter to mark an invalid routine as valid. When a process defines the RDMS$VALIDATE_ROUTINE or RDB_VALIDATE_ROUTINE to 1, Oracle Rdb marks each invalid routine as valid when the process calls the procedure within a read/write transaction.

Digital UNIX

On Digital UNIX, you can define the RDB_VALIDATE_ROUTINE configuration parameter to 1 by including the line in Example A–42 in your .dbsrc configuration file.

Example A–42 Using the RDB_VALIDATE_ROUTINE Configuration Parameter

```
RDB_VALIDATE_ROUTINE 1
```

♦

A.104 RDO$EDIT

OpenVMS VAX OpenVMS Alpha

The RDO$EDIT logical name indicates the system editor selected to edit interactive RDO queries.

Two values are currently allowed:

Logical Value	Selected Editor
EDT	EDIT/EDT
TPU	TPU

The logical can be defined at the system, group, or process level. ♦

A.105 RDOINI

OpenVMS VAX OpenVMS Alpha

The RDOINI logical name specifies the name of the file that contains the RDO initialization information. If the logical name exists, *and* the indicated file exists, RDO executes the commands in this file first, *before* displaying the RDO prompt and accepting input commands.

The logical name is translated using the LNM$FILE_DEV table or tables. ♦

A.106 RMU$EDIT

OpenVMS VAX OpenVMS Alpha

The RMU$EDIT logical name indicates the system editor to be used to edit the notepad in the Performance Monitor tools facility. The valid values are EDT, LSE, and TPU. The default editor is EDT. ♦

A.107 SQL$DATABASE and SQL_DATABASE

The SQL$DATABASE logical name and the SQL_DATABASE configuration parameter specify the database that SQL declares if you do not explicitly declare a database.

Defining SQL$DATABASE or SQL_DATABASE provides a database file specification that interactive SQL automatically uses to find your default database, unless you explicitly attach to any database before you enter a statement that requires database access.

Digital UNIX

Example A–43 shows that using SQL_DATABASE allows you to access your database without explicitly attaching to it.

Example A–43 Using the SQL_DATABASE Configuration Parameter to Define a Default Database

```
SQL_DATABASE /usr/tmp/mf_personnel
$ SQL
SQL> SHOW TABLES
User tables in database with filename /usr/tmp/mf_personnel
     CANDIDATES
     COLLEGES
     CURRENT_INFO                    A view.
     CURRENT_JOB                     A view.
     CURRENT_SALARY                  A view.
     DEGREES
```

(continued on next page)

Example A–43 (Cont.) Using the SQL_DATABASE Configuration Parameter to Define a Default Database

```
     DEPARTMENTS
     EMPLOYEES
     JOBS
     JOB_HISTORY
     RESUMES
     SALARY_HISTORY
     WORK_STATUS
SQL>
```

♦

The SQL precompiler, module processor, and run-time system also translate SQL$DATABASE and SQL_DATABASE if you do not explicitly declare an alias (DECLARE ALIAS) or attach to a database (ATTACH) in a source file or (for precompiled programs) SQL context file.

OpenVMS VAX OpenVMS Alpha

You cannot specify repository access using the SQL$DATABASE logical name. Therefore, if you intend to execute CREATE or ALTER statements including the DICTIONARY IS REQUIRED clause, you should not attach to a database using the SQL$DATABASE logical name. ♦

A.108 SQL$DISABLE_CONTEXT

OpenVMS VAX OpenVMS Alpha

You can disable the two-phase commit protocol by defining the SQL$DISABLE_CONTEXT logical name to be TRUE, as shown in Example A–44

Example A–44 Using the SQL$DISABLE_CONTEXT Logical Name

```
$ DEFINE SQL$DISABLE_CONTEXT TRUE
```

This logical name is useful for turning off distributed transactions when you want to run batch-update transactions. Because batch-update transactions do not write to recovery-unit journal (.ruj) files, these transactions cannot be rolled back and, therefore, cannot be used in a distributed transaction. For more information, see the *Oracle Rdb7 Guide to Distributed Transactions*. ♦

A.109 SQL$EDIT

OpenVMS VAX OpenVMS Alpha

The SQL$EDIT logical name indicates the system editor selected to edit interactive SQL queries.

Two values are currently allowed:

Logical Value	Selected Editor
EDT	EDIT/EDT
TPU	TPU

The logical name can be defined at the system, group, or process level. ♦

A.110 SQLINI

OpenVMS VAX OpenVMS Alpha

The SQLINI logical name specifies the name of the file that contains the SQL initialization information. If the logical name exists, *and* the indicated file exists, SQL executes the commands in this file first, *before* displaying the SQL prompt and accepting input commands.

The logical name is translated using the LNM$FILE_DEV table or tables. ♦

A.111 SQL$KEEP_PREP_FILES and SQL_KEEP_PREP_FILES

You can use the SQL$KEEP_PREP_FILES logical name or the SQL_KEEP_PREP_FILES configuration parameter to cause the SQL precompiler and SQL module language compiler to retain the intermediate .mar and language files. This can be helpful when you are trying to debug an SQL host language module and need to refer to the language files.

Digital UNIX

On Digital UNIX, include the line shown in Example A–45 in your configuration file to specify that you want to retain these files.

Example A–45 Using the SQL_KEEP_PREP_FILES Configuration File

```
SQL_KEEP_PREP_FILES YES
```

♦

B
Oracle Rdb Event-Based Data Tables

OpenVMS VAX OpenVMS Alpha

This appendix describes the event-based data tables in the formatted database for the Oracle Rdb PERFORMANCE and RDBEXPERT collection classes. The ALL collection class is not shown because, for the current release of Oracle Rdb, the ALL class contains the same tables and columns as the RDBEXPERT class.

B.1 Oracle Rdb PERFORMANCE Class Database Tables

This section describes the event-based data tables in the formatted database for the Oracle Rdb PERFORMANCE class. This information is provided so that you can write customized reports based on data in the formatted database.

Table B–1 shows the DATABASE table.

Table B–1 Columns for Table EPC$1_221_DATABASE

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_POINT	DATE VMS	
STREAM_ID	INTEGER	
CLIENT_PC	INTEGER	
DB_NAME	VARCHAR(255)	
DB_NAME_STR_ID	INTEGER	STR_ID_DOMAIN
IMAGE_NAME	VARCHAR(255)	
IMAGE_NAME_STR_ID	INTEGER	STR_ID_DOMAIN

Table B–2 shows the DATABASE_ST table. An index is provided for this table. It is defined with column STR_ID, duplicates are allowed, and the type is sorted.

Table B–2 Columns for Table EPC$1_221_DATABASE_ST

Column Name	Data Type	Domain
STR_ID	INTEGER	STR_ID_DOMAIN
SEGMENT_NUMBER	SMALLINT	SEGMENT_NUMBER_DOMAIN
STR_SEGMENT	VARCHAR(0)	

Table B–3 shows the TRANSACTION table.

Table B–3 Columns for Table EPC$1_221_TRANSACTION

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_START	DATE VMS	
TIMESTAMP_END	DATE VMS	
STREAM_ID_START	INTEGER	
CLIENT_PC_START	INTEGER	
LOCK_MODE_START	SMALLINT	
TRANS_ID_START	VARCHAR(16)	
TRANS_ID_START_STR_ID	INTEGER	STR_ID_DOMAIN
GLOBAL_TID_START	VARCHAR(16)	
GLOBAL_TID_START_STR_ID	INTEGER	STR_ID_DOMAIN
DBS_READS_START	INTEGER	
DBS_WRITES_START	INTEGER	
RUJ_READS_START	INTEGER	
RUJ_WRITES_START	INTEGER	

(continued on next page)

Table B–3 (Cont.) Columns for Table EPC$1_221_TRANSACTION

Column Name	Data Type	Domain
AIJ_WRITES_START	INTEGER	
ROOT_READS_START	INTEGER	
ROOT_WRITES_START	INTEGER	
BUFFER_READS_START	INTEGER	
GET_VM_BYTES_START	INTEGER	
FREE_VM_BYTES_START	INTEGER	
LOCK_REQS_START	INTEGER	
REQ_NOT_QUEUED_START	INTEGER	
REQ_STALLS_START	INTEGER	
REQ_DEADLOCKS_START	INTEGER	
PROM_DEADLOCKS_START	INTEGER	
LOCK_RELS_START	INTEGER	
LOCK_STALL_TIME_START	INTEGER	
BIO_START	INTEGER	
DIO_START	INTEGER	
PAGEFAULTS_START	INTEGER	
PAGEFAULT_IO_START	INTEGER	
CPU_START	INTEGER	
CURRENT_PRIO_START	SMALLINT	
VIRTUAL_SIZE_START	INTEGER	
WS_SIZE_START	INTEGER	
WS_PRIVATE_START	INTEGER	
WS_GLOBAL_START	INTEGER	
DBS_READS_END	INTEGER	
DBS_WRITES_END	INTEGER	
RUJ_READS_END	INTEGER	
RUJ_WRITES_END	INTEGER	
AIJ_WRITES_END	INTEGER	
ROOT_READS_END	INTEGER	

(continued on next page)

Table B–3 (Cont.) Columns for Table EPC$1_221_TRANSACTION

Column Name	Data Type	Domain
ROOT_WRITES_END	INTEGER	
BUFFER_READS_END	INTEGER	
GET_VM_BYTES_END	INTEGER	
FREE_VM_BYTES_END	INTEGER	
LOCK_REQS_END	INTEGER	
REQ_NOT_QUEUED_END	INTEGER	
REQ_STALLS_END	INTEGER	
REQ_DEADLOCKS_END	INTEGER	
PROM_DEADLOCKS_END	INTEGER	
LOCK_RELS_END	INTEGER	
LOCK_STALL_TIME_END	INTEGER	
BIO_END	INTEGER	
DIO_END	INTEGER	
PAGEFAULTS_END	INTEGER	
PAGEFAULT_IO_END	INTEGER	
CPU_END	INTEGER	
CURRENT_PRIO_END	SMALLINT	
VIRTUAL_SIZE_END	INTEGER	
WS_SIZE_END	INTEGER	
WS_PRIVATE_END	INTEGER	
WS_GLOBAL_END	INTEGER	
RDB_CROSS_FAC_2	INTEGER	
RDB_CROSS_FAC_3	INTEGER	
RDB_CROSS_FAC_7	INTEGER	
RDB_CROSS_FAC_14	INTEGER	

Table B–4 shows the TRANSACTION_ST table. An index is provided for this table. It is defined with column STR_ID, duplicates are allowed, and the type is sorted.

Table B–4 Columns for Table EPC$1_221_TRANSACTION_ST

Column Name	Data Type	Domain
STR_ID	INTEGER	STR_ID_DOMAIN
SEGMENT_NUMBER	SMALLINT	SEGMENT_NUMBER_DOMAIN
STR_SEGMENT	VARCHAR(0)	

Table B–5 shows the REQUEST_ACTUAL table.

Table B–5 Columns for Table EPC$1_221_REQUEST_ACTUAL

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_START	DATE VMS	
TIMESTAMP_END	DATE VMS	
DBS_READS_START	INTEGER	
DBS_WRITES_START	INTEGER	
RUJ_READS_START	INTEGER	
RUJ_WRITES_START	INTEGER	
AIJ_WRITES_START	INTEGER	
ROOT_READS_START	INTEGER	
ROOT_WRITES_START	INTEGER	
BUFFER_READS_START	INTEGER	
GET_VM_BYTES_START	INTEGER	
FREE_VM_BYTES_START	INTEGER	
LOCK_REQS_START	INTEGER	
REQ_NOT_QUEUED_START	INTEGER	
REQ_STALLS_START	INTEGER	
REQ_DEADLOCKS_START	INTEGER	

(continued on next page)

Table B–5 (Cont.) Columns for Table EPC$1_221_REQUEST_ACTUAL

Column Name	Data Type	Domain
PROM_DEADLOCKS_START	INTEGER	
LOCK_RELS_START	INTEGER	
LOCK_STALL_TIME_START	INTEGER	
BIO_START	INTEGER	
DIO_START	INTEGER	
PAGEFAULTS_START	INTEGER	
PAGEFAULT_IO_START	INTEGER	
CPU_START	INTEGER	
CURRENT_PRIO_START	SMALLINT	
VIRTUAL_SIZE_START	INTEGER	
WS_SIZE_START	INTEGER	
WS_PRIVATE_START	INTEGER	
WS_GLOBAL_START	INTEGER	
STREAM_ID_END	INTEGER	
CLIENT_PC_END	INTEGER	
REQ_ID_END	INTEGER	
COMP_STATUS_END	INTEGER	
REQUEST_OPER_END	INTEGER	
TRANS_ID_END	VARCHAR(16)	
TRANS_ID_END_STR_ID	INTEGER	STR_ID_DOMAIN
DBS_READS_END	INTEGER	
DBS_WRITES_END	INTEGER	
RUJ_READS_END	INTEGER	
RUJ_WRITES_END	INTEGER	
AIJ_WRITES_END	INTEGER	
ROOT_READS_END	INTEGER	
ROOT_WRITES_END	INTEGER	
BUFFER_READS_END	INTEGER	
GET_VM_BYTES_END	INTEGER	

(continued on next page)

Table B–5 (Cont.) Columns for Table EPC$1_221_REQUEST_ACTUAL

Column Name	Data Type	Domain
FREE_VM_BYTES_END	INTEGER	
LOCK_REQS_END	INTEGER	
REQ_NOT_QUEUED_END	INTEGER	
REQ_STALLS_END	INTEGER	
REQ_DEADLOCKS_END	INTEGER	
PROM_DEADLOCKS_END	INTEGER	
LOCK_RELS_END	INTEGER	
LOCK_STALL_TIME_END	INTEGER	
BIO_END	INTEGER	
DIO_END	INTEGER	
PAGEFAULTS_END	INTEGER	
PAGEFAULT_IO_END	INTEGER	
CPU_END	INTEGER	
CURRENT_PRIO_END	SMALLINT	
VIRTUAL_SIZE_END	INTEGER	
WS_SIZE_END	INTEGER	
WS_PRIVATE_END	INTEGER	
WS_GLOBAL_END	INTEGER	

Table B–6 shows the REQUEST_ACTUAL_ST table. An index is provided for this table. It is defined with column STR_ID, duplicates are allowed, and the type is sorted.

Table B–6 Columns for Table EPC$1_221_REQUEST_ACTUAL_ST

Column Name	Data Type	Domain
STR_ID	INTEGER	STR_ID_DOMAIN
SEGMENT_NUMBER	SMALLINT	SEGMENT_NUMBER_ DOMAIN
STR_SEGMENT	VARCHAR(0)	

B.2 Oracle Rdb RDBEXPERT Class Database Tables

This section describes the event-based data tables in the formatted database for the Oracle Rdb RDBEXPERT collection class. This information is provided so that you can write customized reports based on data in the formatted database.

The RDBEXPERT collection class contains all the event-based data tables shown in Section B.1, plus the two tables shown in Table B–7 and Table B–8.

Table B–7 shows the REQUEST_BLR table.

Table B–7 Columns for Table EPC$1_221_REQUEST_BLR

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_POINT	DATE VMS	
STREAM_ID	INTEGER	
TRANS_ID	INTEGER	
CLIENT_PC	INTEGER	
REQ_ID	INTEGER	
BLR	VARCHAR(127)	
BLR_STR_ID	INTEGER	STR_ID_DOMAIN

Table B–8 shows the REQUEST_BLR_ST table. An index is provided for this table. It is defined with column STR_ID, duplicates are allowed, and the type is sorted.

Table B–8 Columns for Table EPC$1_221_REQUEST_BLR_ST

Column Name	Data Type	Domain
STR_ID	INTEGER	STR_ID_DOMAIN
SEGMENT_NUMBER	SMALLINT	SEGMENT_NUMBER_DOMAIN

(continued on next page)

Table B–8 (Cont.) Columns for Table EPC$1_221_REQUEST_BLR_ST

Column Name	Data Type	Domain
STR_SEGMENT	VARCHAR(128)	

♦

C

Using RDMS$DEBUG_FLAGS and RDB_DEBUG_FLAGS to Analyze the Query Optimizer

Oracle Rdb enables you to examine the access strategies selected by the query optimizer and the estimated cost of those strategies at query execution time by defining the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter. By analyzing the displays from this feature, you can develop indexing policies for routine queries, as well as for those queries that contain complex record selection expressions.

OpenVMS VAX OpenVMS Alpha

You can define the RDMS$DEBUG_FLAGS logical name as one or more flags, for example:

```
$ DEFINE RDMS$DEBUG_FLAGS  "S"

     or

$ DEFINE RDMS$DEBUG_FLAGS  "SO"
```

♦

Digital UNIX

You can define the RDB_DEBUG_FLAGS configuration parameter as one or more flags in the your .dbsrc configuration file. For example:

```
RDB_DEBUG_FLAGS  "S"

     or

RDB_DEBUG_FLAGS  "SO"
```

♦

Table C–1 shows the available flags, the output generated, and where to find more information.

Table C–1 Flags Used with the RDMS$DEBUG_FLAGS Logical Name and the RDB_DEBUG_FLAGS Configuration Parameter

Flag	Output generated	Reference
E	Execution trace	Section C.6
O	Query statistics	Section C.4, Section C.5
R	Sort statistics	Section C.7
S	Query strategy	Section C.1, Section C.5, Section C.6
Ss	Query outline definitions for nonsystem queries	Section C.2
ISs	Query outline definitions for system-created queries	Section C.2
ISsn	Constraint and/or trigger names for which query outlines are generated	Section C.2
Sn	Constraints query strategy	Section C.3
Xt	TRACE control statement logging	Section C.9
T	Transaction activity	Section C.8
\	Sets dbkey buffer size to 10	Section C.6

The flags that can be defined for RDMS$DEBUG_FLAGS and RDB_DEBUG_FLAGS are case sensitive. Therefore, when you define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter as one or more flags, be sure to specify each flag in the correct case (uppercase or lowercase). If, for example, you want to display constraint names with a query strategy, be sure to specify the flags as "Sn" (uppercase S and lowercase n). If you use the wrong case for a flag or flags, the output produced by the flags will be different than what is documented in this chapter.

The RDMS$DEBUG_FLAGS_OUTPUT logical name and the RDB_DEBUG_FLAGS_OUTPUT configuration parameter allow you to specify a file in which to collect the output from RDMS$DEBUG_FLAGS and RDB_DEBUG_FLAGS_OUTPUT, respectively. You must have write access to the target directory for this to be successful.

Digital UNIX

For example, you could define the RDB_DEBUG_FLAGS and RDB_DEBUG_FLAGS_OUTPUT configuration parameters in your .dbsrc configuration file as in the following example:

```
RDB_DEBUG_FLAGS "S"
RDB_DEBUG_FLAGS_OUTPUT debugflags_output.log
```

In this case, Oracle Rdb writes all the strategy information generated by the S flag to the file debugflags_output.log in your default directory. ♦

- If you do not define RDMS$DEBUG_FLAGS_OUTPUT or RDB_DEBUG_FLAGS_OUTPUT, output is directed to the default device. The default device on OpenVMS is SYS$OUTPUT and the default device on Digital UNIX is the standard output (stdout) device.
- If you do not specify a file type when you define RDMS$DEBUG_FLAGS_OUTPUT or RDB_DEBUG_FLAGS_OUTPUT, the file receives the default file type of .lis.
- If you do direct RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS_OUTPUT output to a file, the query and its results do not appear in the file. If output is displayed on the default device, Oracle Rdb shows the query, the output from RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS_OUTPUT, and the query results.

Note

You must define the RDMS$DEBUG_FLAGS and RDMS$DEBUG_FLAGS_OUTPUT logical names or the RDB_DEBUG_FLAGS and RDB_DEBUG_FLAGS_OUTPUT configuration parameters before you attach to your database. Once attached, you cannot reset the logical names or configuration parameters without first detaching from the database.

You can use the SQL SET NOEXECUTE statement to control query execution. By using the NOEXECUTE option in conjunction with the S and O flags of RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS, you can examine the access strategy and estimated cost of a query, without actually executing the query or displaying results. This can be a valuable tool if you are testing various queries to compare access strategies or costs, but are not concerned with query output.

C.1 Displaying Optimization Strategy with the S Flag

When you define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter as S and execute a query, Oracle Rdb returns a formatted display that shows the access method or methods the optimizer used to produce the results of a query. Table C–2 shows the notation that can appear in the S flag display.

Table C–2 Output Definitions for the S Flag

Output	Definition
Aggregate	Indicates use of a statistical function, such as COUNT, SUM, AVG, MIN, MAX, or GROUP BY.
Aggregate-F1	Indicates a check for the existence of a single value, such as a query containing EXIST or ANY.
Aggregate-F2	Indicates a check for uniqueness. Used when a query contains UNIQUE.
BgrOnly	Indicates the background only leaf retrieval type.
Bool	Indicates key-only Boolean optimization. The optimizer uses this method to filter out dbkeys before fetching rows, thus saving I/O operations.
Card=*n*	Indicates table cardinality stored in the field RDB$CARDINALITY in the system relation RDB$RELATIONS.
Conjunct	Indicates processing of a WHERE predicate. Also means that the optimizer could not use an index to (fully) satisfy the expression from the query. Oracle Rdb can process several WHERE constructs as part of one Conjunct. Therefore, the number of occurrences of the word Conjunct does not necessarily reflect the number of comparisons involved in the query.
Cross block of *n* entries	Indicates a cross (or nested loop) join method for n entries.
Direct lookup	Indicates that the index used has no duplicates and that an exact key match predicate is used for retrieval, which returns one or zero dbkeys.
Fan=*n*	Indicates the average fanout factor of a B-tree node based on the B-tree node size, index key length, and the initial percent fill of the index node. The fanout of a B-tree node is the number of child nodes (branches) attached to a given node. The higher the average fanout, the fewer levels the B-tree contains, which promotes faster single key access but greater potential for deadlock in a multiuser environment.
FFirst	Indicates the fast first leaf retrieval type.

(continued on next page)

Table C–2 (Cont.) Output Definitions for the S Flag

Output	Definition
Firstn	Indicates use of a LIMIT TO n ROWS clause. This information is not used by the optimizer as part of the optimization process, but rather as a limitation of the output from the query. The processing of the Firstn clause is always the last thing done and is therefore always shown as the first line of output.
Get	Indicates execution of an I/O operation for data record retrieval.
Index only retrieval	Indicates the requested information was retrieved from within the index. No data record access was required.
Leaf#01	Identifies the first leaf node in the execution tree. Subsequent leaf nodes in the same tree are incremented by one.
Match	Indicates a match (or merge scan) join method.
Max key lookup Min key lookup	Indicates a direct index key lookup instead of an entire index scan. Used when the optimizer can use index only retrieval and the query contains the MAX or MIN statistical function. The optimizer uses the Max or Min aggregate optimization under the following circumstances: • Used on ascending, descending, and partitioned indexes. • For a multisegmented index, the optimizer can perform MAX or MIN only on the leading segment unless a trailing segment is specified and the leading segment is an equality. • Not possible if the query contains a nonindexed predicate. • Not possible if any indexed predicate is not used during index lookup.
Merge block of *n* entries	Indicates use of the merge strategy to return rows from multiple tables. In the merge strategy, rows from different tables are concatenated and delivered to the user.
NdxOnly	Indicates the index only leaf retrieval type.
OR index retrieval	Indicates static OR optimization using a hashed index or multiple sorted indexes to retrieve data from a single table. Rows selected by each index are delivered one after another. Duplicate dbkeys are discarded by applying Conjunct, so that the same row is not delivered twice.
Reduce	Indicates elimination of duplicate rows based on the values of one or more columns. Found in queries using the DISTINCT or UNION operators. A sort is frequently part of the duplicate elimination process and the Sort notation is therefore often included in the output.

(continued on next page)

Table C–2 (Cont.) Output Definitions for the S Flag

Output	Definition
Retrieval by dbk of relation	Indicates that the requested data was retrieved using direct dbkey access.
Retrieval by index of relation	Indicates use of an index to retrieve data.
Retrieval sequentially of relation	Indicates the table or relation is read sequentially.
Sort	Indicates the requested data had an output order specified, that a sort was done on behalf of a match join strategy, or that a sort was required for a Reduce operation.
Sorted	Indicates the sorted order leaf retrieval type.
Temporary relation	Indicates creation of a temporary table to store intermediate results. The temporary table can exist either in memory or on disk. See Section 3.2.1.7 for information on using the RDMS$BIND_WORK_VM and RDMS$BIND_WORK_FILE logical names or the RDB_BIND_WORK_VM and RDB_BIND_WORK_FILE configuration parameters to reduce the disk I/O that can accompany the creation of a temporary table.
Zigzag	Indicates a variation of the match retrieval strategy.
[l:h]	Indicates the number of low index key (low Ikey) segments and high index key (high Ikey) segments in an index key range. *l* represents the number of low Ikey segments and *h* represents the number of high Ikey segments for the given index range. • The notation [0:0] indicates that a full scan of the index is done. • The notation [1:1] indicates there is one low index key segment and one high index key segment in the index key range. Sometimes this indicates the presence of an equality predicate in the query, such as EMP_EMPLOYEE_ID = '00164'. In this case, the range is a single value. • The notation [1:0] indicates there is one low index key segment but no high index key segment. That is, the range of index keys to be scanned has a starting point but no upper limit. • The notation [0:1] indicates there is no low index key segment but one high index key segment. That is, the range of index keys to be scanned starts at the beginning of the index but ends at the high index key bound.

(continued on next page)

Table C–2 (Cont.) Output Definitions for the S Flag

Output	Definition

[l:h . . .]n	Indicates dynamic OR optimization in which the optimizer uses a single sorted index to locate two or more ranges of data rows. *l* represents the number of low Ikey segments in the first range; *h* represents the number of high Ikey segments in the first range; . . . represents subsequent index ranges; and, *n* represents the total number of ranges. For example:
	`.`
	`.`
	`.`
	`WHERE EMPLOYEE_ID IN (’00164’, ’00177’, ’00200’);`
	`Leaf#01 FFirst R Card=100`
	`  BgrNdx1 EMP_EMPLOYEE_ID [1:1...]3`
	`.`
	`.`
	`.`
	In this case, the three ranges correspond to the three employee ID numbers.

~S#0001	Indicates that the following lines show the strategy for query number 0001.

The remainder of this section provides examples of S flag output that illustrate various access strategies. Other examples of S flag output are shown in Section C.5 and Section C.6.

Sequential Access Strategy

Example C–1 shows simple sequential access when RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as “S”.

Example C–1 S Display—Sequential Access

```
SQL> SELECT LAST_NAME, FIRST_NAME
cont> FROM CANDIDATES
cont> ORDER BY LAST_NAME;
Sort ❸   Get ❷     Retrieval sequentially of relation CANDIDATES ❶
 LAST_NAME        FIRST_NAME
 Boswick          Fred
```

(continued on next page)

Example C–1 (Cont.) S Display—Sequential Access

```
 Schwartz        Trixie
 Wilson          Oscar
3 rows selected
SQL>
```

The following callouts are keyed to Example C–1:

❶ The optimizer selects a sequential retrieval strategy because all rows must be returned from the table CANDIDATES.

❷ Indicates data record fetches for the requested rows.

❸ Indicates the requested rows have been sorted as specified by the ORDER BY clause.

To improve performance of a query, pay particular attention to the notation, “Retrieval sequentially.” You may be able to improve performance if you define an index for the column involved. You should not assume, however, that simply defining indexes automatically speeds up a query, or that Oracle Rdb uses the index you have defined. The correct procedure is to define the index, run the query with RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS again, and check the output for the notation, “Indexed retrieval.” You should time the query to determine if adding an index really improves performance. In some cases, Oracle Rdb ignores the index because it is faster to access rows sequentially. Also, check the cost of a query by setting the O flag to determine which query would result in fewer I/O operations.

In addition, you should note the presence of the sort notation. Sorting rows before delivery usually slows down query execution time. By defining the appropriate index, you can prevent row sorting, thus speeding up query execution time.

Indexed Access Strategy

Example C–2 shows an indexed access strategy when RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as “S”.

Example C–2 S Display—Indexed Access

```
SQL> SELECT LAST_NAME FROM EMPLOYEES
cont> WHERE EMPLOYEE_ID = ’00167’;
```

(continued on next page)

Example C–2 (Cont.) S Display—Indexed Access

```
Get ❺   Retrieval by Index  of relation EMPLOYEES ❶
  Index name EMPLOYEES_HASH ❷ [1:1] ❸  Direct lookup ❹
 LAST_NAME
 Kilpatrick
1 row selected
SQL>
```

The following callouts are keyed to Example C–2:

❶ The optimizer determines that indexed retrieval is the best strategy.

❷ The EMPLOYEES_HASH hashed index is selected.

❸ Indicates the index range to scan, in this case, one low index key segment and one high index key segment. For hashed indexes these values are the same because the range is limited to a single value.

❹ The index does not allow duplicates and will return one or zero dbkeys.

❺ The index itself does not contain all columns required by the query, so the optimizer must fetch the data row (indicates data I/O).

Index Only Access Strategy

Example C–3 illustrates the index only access strategy when RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as “S”.

Example C–3 S Display—Index Only Access

```
SQL> SELECT LAST_NAME FROM EMPLOYEES
cont> WHERE LAST_NAME STARTING WITH 'A';
Conjunct ❺       Index only retrieval of relation EMPLOYEES ❶ ❹
  Index name  EMP_LAST_NAME ❷ [1:1] ❸
 LAST_NAME
 Ames
 Andriola
2 rows selected
SQL>
```

The following callouts are keyed to Example C–3:

❶ The optimizer determines that index only retrieval is the best strategy.

❷ The EMP_LAST_NAME index is selected.

❸ Indicates the index range to scan; in this case, one low index key segment and one high index key segment.

❹ Because only the LAST_NAME column is required to satisfy the query and the EMP_LAST_NAME index is based on that column, the index itself contains all the required data.

❺ Returned index data is tested to see if the data matches the condition specified in the WHERE clause.

Index Access Strategy with OR

Example C–4 illustrates traditional (static) OR index retrieval strategy. Compare Example C–4 with Example C–5, which illustrates dynamic OR optimization.

Example C–4 S Display—OR Indexed Retrieval

```
SQL> SELECT EMPLOYEE_ID, LAST_NAME
cont> FROM EMPLOYEES
cont> WHERE EMPLOYEE_ID IN ('00167', '00168');
OR index retrieval ❶
  Get     Retrieval by index of relation EMPLOYEES ❸
    Index name  EMPLOYEES_HASH [1:1]       Direct lookup ❷
  Conjunct        Get     Retrieval by index of relation EMPLOYEES ❺
    Index name  EMPLOYEES_HASH [1:1]       Direct lookup ❹
 EMPLOYEE_ID   LAST_NAME ❻
 00167         Kilpatrick
 00168         Nash
2 rows selected
SQL>
```

The following callouts are keyed to Example C–4:

❶ The optimizer selects OR index retrieval because the query contains the IN operator and a hashed index exists on the EMPLOYEE_ID column.

❷ In the first leg of OR indexed retrieval, the optimizer uses the index EMPLOYEES_HASH to search for the specified EMPLOYEE_ID (00167) without scanning the index (Direct lookup). The optimizer returns a single dbkey.

❸ The optimizer uses the returned dbkey to retrieve (Get) the row from the EMPLOYEES table.

❹ In the second leg of OR indexed retrieval, the optimizer uses the index EMPLOYEES_HASH to search for the next EMPLOYEE_ID (00168) without scanning the index (Direct lookup). The optimizer returns a single dbkey.

❺ The optimizer uses the returned dbkey to retrieve (Get) the row from the EMPLOYEES table. The retrieved row is compared with previously returned rows (Conjunct). If the row is different, it is delivered; if the row is a duplicate, it is discarded.

❻ All the retrieved rows are delivered to the user.

Index Access Strategy with Dynamic OR Optimization

Example C–5 illustrates dynamic OR index retrieval strategy. Compare Example C–5 with Example C–4. Both examples use the same query but Example C–5 uses the personnel database; Example C–4 uses the mf_personnel database. Because hashed indexes are not available on single-file databases, Example C–5 uses a sorted index to process the two key ranges.

Example C–5 S Display—Dynamic OR Indexed Retrieval

```
SQL> SELECT EMPLOYEE_ID, LAST_NAME
cont> FROM EMPLOYEES
cont> WHERE EMPLOYEE_ID IN (’00167’, ’00168’);
Leaf#01 FFirst EMPLOYEES Card=100 ❶
  BgrNdx1 EMP_EMPLOYEE_ID [1:1...]2 Fan=17 ❷
 EMPLOYEE_ID   LAST_NAME ❸
 00167         Kilpatrick
 00168         Nash
2 rows selected
SQL>
```

The following callouts are keyed to Example C–5:

❶ The optimizer uses the fast first (FFirst) type of dynamic leaf retrieval on the EMPLOYEES table, which contains 100 rows. Leaf#01 indicates that this is the first (and only) node in the execution tree.

❷ The second line of output has the following elements:

- EMP_EMPLOYEE_ID is the best index available and thus is the only index (BgrNdx1) the optimizer will use to retrieve data rows. Although not shown in the display, the foreground process uses the same index and actually delivers the rows.
- [1:1 . . .]2 is the notation that reveals this strategy as dynamic OR optimization. There is one low index key segment and one high index key segment for the first range (00167) of the EMP_EMPLOYEE_ID index. The number “2” indicates the number of ranges; in this case, employee IDs 00167 and 00168.

- Fan=17 indicates a fanout factor of 17, meaning that there are an average of 17 branches in the B-tree representing this index.

❸ The retrieved rows are delivered to the user.

Cross Retrieval Strategy

Example C–6 illustrates a join of two tables using the cross block retrieval strategy when RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is defined as “S”.

Example C–6 S Display—Cross Block Retrieval

```
SQL> SELECT DISTINCT JH.EMPLOYEE_ID, J.JOB_TITLE
cont> FROM JOB_HISTORY JH, JOBS J
cont> WHERE JH.JOB_CODE = J.JOB_CODE;
Reduce  Sort ❻
Cross block of 2 entries ❶
  Cross block entry 1 ❷
    Get     Retrieval sequentially of relation JOBS ❸
  Cross block entry 2 ❹
    Conjunct        Get     Retrieval sequentially of relation JOB_HISTORY ❺
 JH.EMPLOYEE_ID   J.JOB_TITLE
 00164            Department Manager
 00164            Systems Programmer
 00165            Assistant Clerk
   .
   .
   .
 00435            Vice President
 00471            Department Manager
193 rows selected
SQL>
```

The following callouts are keyed to Example C–6:

❶ The optimizer uses the cross block retrieval strategy to access the data in the JOBS table (entry 1) and the JOB_HISTORY table (entry 2).

❷ The optimizer starts processing with Cross block entry 1. Each cross block entry is read from top to bottom.

❸ Data rows in the JOBS table are accessed sequentially. The Get notation represents a data read and indicates that the optimizer is fetching a row in the JOBS table.

❹ The optimizer uses cross block entry 2 to process the JOB_HISTORY table.

❺ Data rows in the JOB_HISTORY table are accessed sequentially. The Get notation represents a data read and indicates that the optimizer is fetching a row in the JOB_HISTORY table.

❻ The resulting rows from each table (block) are sorted and duplicates are eliminated.

C.2 Displaying Outlines Generated by the Optimizer with the Ss, ISs, and ISsn Flags

You define an outline by using query outlines generated by the Oracle Rdb optimizer. Defining the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter with the following flag combinations enables display of query outline definitions:

- "Ss" flags–display query outline definitions for all nonsystem queries.
- "ISs" flags–display query outline definitions for system-created queries such as those generated when Oracle Rdb compiles constraint and trigger internal queries.
- "ISsn" flags–display the names of the constraints or triggers (or both) for which query outlines are generated.

OpenVMS VAX OpenVMS Alpha For example, the following command combines the "I" (Internals) flag with the "Ss" (Dump query outlines) flags:

```
$ DEFINE RDMS$DEBUG_FLAGS "ISs"
```

The following command adds the "n" (Show names) flag to the flags in the preceding example:

```
$ DEFINE RDMS$DEBUG_FLAGS "ISsn"
```

♦

Note that the "s" and "n" flags must be lowercase and must immediately follow the uppercase "S" (Strategy) flag.

See Section 5.9.1 for complete information on capturing an outline generated by the optimizer, interpreting the generated outline, and editing the generated outline before storing it in the database. Example 5–6 shows how to capture outlines generated by the optimizer.

C.3 Displaying Constraint Names and the Query Strategy with the Sn Flag

When you define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter as Sn, Oracle Rdb displays the names of constraints that were forced to be evaluated by your queries, along with the constraints query strategy. Note that you must use an uppercase S and a lowercase n when you define RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS as Sn. Example C–7 shows the Sn flag display.

Example C–7 Sn Flag Display

```
SQL> INSERT INTO EMPLOYEES
cont> (EMPLOYEE_ID,
cont>  LAST_NAME,
cont>  FIRST_NAME,
cont>  MIDDLE_INITIAL,
cont>  ADDRESS_DATA_1,
cont>  ADDRESS_DATA_2,
cont>  CITY,
cont>  STATE,
cont>  POSTAL_CODE,
cont>  SEX,
cont>  BIRTHDAY,
cont>  STATUS_CODE)
cont> VALUES
cont> (’98765’,
cont>  ’Snerd’,
cont>  ’Mortimer’,
cont>  ’J’,
cont>  ’149 Pothole Place’,
cont>  ’Apartment 7’,
cont>  ’Nashua’,
cont>  ’NH’,
cont>  ’03060’,
cont>  ’M’,
cont>  ’07-Dec-1993’,
cont>  ’1’);
1 row inserted
SQL> COMMIT;
```

(continued on next page)

Example C–7 (Cont.) Sn Flag Display

```
~S: Constraint name  EMPLOYEES_PRIMARY_EMPLOYEE_ID  ❶
Cross block of 2 entries
  Cross block entry 1
    Conjunct        Firstn  Get     Retrieval by DBK of relation EMPLOYEES
  Cross block entry 2
    Conjunct        Aggregate-F2    Get
    Retrieval by index of relation EMPLOYEES
      Index name  EMPLOYEES_HASH [1:1]       Direct lookup
~S: Constraint name  EMP_SEX_VALUES    Conjunct        Firstn  Get  ❷
Retrieval by DBK of relation EMPLOYEES
~S: Constraint name  EMP_STATUS_CODE_VALUES    Conjunct        Firstn  Get ❸
Retrieval by DBK of relation EMPLOYEES
SQL>
```

The following callouts are keyed to Example C–7:

❶ The primary key constraint EMPLOYEES_PRIMARY_EMPLOYEE_ID on the EMPLOYEE_ID column of the EMPLOYEES table is evaluated by the insertion of the employee record.

❷ The check constraint EMP_SEX_VALUES on the SEX column of the EMPLOYEES table is evaluated by the insertion of the employee record.

❸ The check constraint EMP_STATUS_CODE_VALUES on the STATUS_ CODE column of the EMPLOYEES table is evaluated by the insertion of the employee record.

Because the three constraints referenced by the request in Example C–7 are all evaluated at commit time, the constraints are not compiled and the names and strategy are not displayed until the COMMIT statement is issued.

C.4 Displaying Optimization Statistics with the O Flag

When you define the RDMS$DEBUG_FLAGS logical name or the RDB_ DEBUG_FLAGS configuration parameter as O, Oracle Rdb displays statistics that show the number of solutions the query optimizer tried and rejected before it found an optimal solution for the query. It also displays the estimated cost for executing the query; that is, the expected number of rows in the query result. A sample display is shown in Example C–8.

Example C–8 O Flag Display

```
SQL> SELECT DISTINCT JH.EMPLOYEE_ID, J.JOB_TITLE
cont> FROM JOB_HISTORY JH, JOBS J
cont> WHERE JH.JOB_CODE = J.JOB_CODE;
Solutions tried 11 ❶
Solutions blocks created 8 ❷
Created solutions pruned 5 ❸
Cost of the chosen solution      1.3184742E+03 ❹
Cardinality of chosen solution   2.4829416E+02 ❺
 JH.EMPLOYEE_ID   J.JOB_TITLE
 00164            Department Manager
 00164            Systems Programmer
 00165            Assistant Clerk
 00166            Associate Programmer
 00166            Department Manager
 00167            Associate Programmer
   .
   .
   .
 00435            Department Manager
 00435            Vice President
 00471            Department Manager
193 rows selected ❻
SQL>
```

The following callouts are keyed to Example C–8:

❶ The number of actual retrieval solutions analyzed by the query optimizer, in this case 11.

❷ The number of retrieval solutions that the query optimizer found *interesting*, in this case 8. Interesting solutions are those with fewer I/O operations or those with a sort order that can be used in another part of the query.

❸ The number of solutions that the optimizer pruned from the solution queue because another, less costly method of retrieval was found, in this case 5.

❹ The *estimated* relative cost of the full query. In this case, 1318 I/O operations.

❺ The *estimated* number of rows returned, in this case 248 rows. The optimizer determines cardinality by estimating the number of rows the query is expected to return based on many factors, including:

- The number of tables and their cardinalities
- The selectivities of the query predicates

❻ The number of rows actually returned. The optimizer estimated a return of 248 rows; the actual number was 193.

This display can be useful when you are developing a very complex query that joins many tables. Note that some overhead included in such a query results from query optimizer processing overhead. You can experiment with different forms of the query by defining indexes for particular columns named in the select expression, or separating the query into smaller and simpler queries. Then you can run the query using the S and O flags and compare the access strategies the query optimizer chooses for each form of the query with the cost. Use the form of the query that displays the lowest relative cost. Usually the best solution is to simplify the query as much as possible.

In general, you should be concerned with query execution cost only if a particular query poses a problem. Because the database can change considerably from one execution of a query to the next, the optimizer may choose a different access strategy for each execution. Special cases involving complex select expressions, however, may benefit from this kind of analysis before you include them in your host language programs.

You can also access optimizer cost information using the interactive SQL statement SET QUERY CONFIRM. See Section 5.8.8.

C.5 Displaying the Optimization Strategy and Cost of Optimization Using the SO Flags

When you define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter as SO, Oracle Rdb displays statistics that show both the number of solutions the query optimizer tried and rejected before performing the query, and the retrieval strategy or strategies that the optimizer used to perform the query.

Example C–9 and Example C–11 use identical queries to demonstrate the advantages or trade-offs of different access strategies.

Example C–9 SO Flag Display

```
SQL> SELECT JOB_CODE, JOB_TITLE, MINIMUM_SALARY, MAXIMUM_SALARY
cont> FROM JOBS
cont> WHERE WAGE_CLASS= '4'
cont> AND MINIMUM_SALARY BETWEEN 10000 AND 60000;
```

(continued on next page)

Example C–9 (Cont.) SO Flag Display

```
Solutions tried 1
Solutions blocks created 1
Created solutions pruned 0
Cost of the chosen solution           9.0000000E+00  ! (9 I/OS)
Cardinality of chosen solution        5.7421874E-02  ! (0.06 rows)
Conjunct        Get     Retrieval sequentially of relation JOBS
 JOB_CODE   JOB_TITLE             MINIMUM_SALARY   MAXIMUM_SALARY
 APGM       Associate Programmer      $15,000.00       $24,000.00
 DMGR       Department Manager        $50,000.00      $100,000.00
 DSUP       Dept. Supervisor          $36,000.00       $60,000.00
 EENG       Electrical Engineer       $20,000.00       $40,000.00
 MENG       Mechanical Engineer       $20,000.00       $35,000.00
 PRGM       Programmer                $20,000.00       $35,000.00
 SANL       Systems Analyst           $40,000.00       $60,000.00
 SPGM       Systems Programmer        $25,000.00       $50,000.00
8 rows selected
SQL>
```

Because no indexes are defined for either column used in this query, the optimizer must search the JOBS table sequentially to retrieve the rows identified in the select expression. The output displayed by the S flag shows that only one access solution was created: a sequential retrieval of the rows in the table. The cost associated with this solution, displayed by the O flag, is determined by the estimated number of I/O operations necessary to retrieve the selected rows, in this case 9. The estimated cardinality value of the chosen solution is 0.06 rows retrieved.

Unless a query includes the ORDER BY clause, the optimizer does not guarantee a specific row order. The optimizer finds all the rows that satisfy the query. In this case, the rows in the JOBS table were stored using the JOB_CODE order, and that order is relatively stable. The optimizer returns rows by ascending JOB_CODE, just as they occur in the table itself.

Note

In these examples, the order of the rows displayed differs from one query to the next. The order depends on which columns, if any, are indexed. When you define an index for a column, the optimizer arranges the nodes in the index in ascending order of value. Therefore, the default sort order for displayed indexed columns is ASCENDING.

If you need to specify a particular order for any column, always include an ORDER BY clause in your query. Because you cannot determine the access method the optimizer chooses to retrieve your rows, do not rely on a default sort order.

Because the JOBS table is small and relatively stable (not likely to grow much if all jobs categories are already defined), sequential access seems to be the fastest access method. Furthermore, small tables should be placed in uniform storage areas to realize the fastest possible access. Sequential access on mixed storage areas incurs a higher search overhead. If the JOBS table were to grow considerably for some reason, you might consider defining an index and testing it again. You should time an actual access to determine if time is saved when using an index. A general rule is that sequential access probably works best for small tables that are stable; a small table is one where there are a small number of unique column values with no duplicates. If you are unsure, define an index and test it as shown in Example C–10.

Adding an index, as shown in Example C–10, enables the optimizer to consider another access strategy to retrieve rows that satisfy the query.

Example C–10 Defining the WAGECLASS_IDX Index

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ WRITE;
SQL> CREATE INDEX WAGECLASS_IDX ON JOBS
cont> (WAGE_CLASS)
cont> TYPE IS SORTED;
SQL> COMMIT;
```

Example C–11 uses the same query shown in Example C–9, but uses the WAGE_CLASS index defined in Example C–10.

Example C–11 SO Flag Display Using the New WAGECLASS_IDX Index

```
SQL> SELECT JOB_CODE, JOB_TITLE, MINIMUM_SALARY, MAXIMUM_SALARY
cont> FROM JOBS
cont> WHERE WAGE_CLASS= '4'
cont> AND MINIMUM_SALARY BETWEEN 10000 AND 60000;
```

(continued on next page)

Example C–11 (Cont.) SO Flag Display Using the New WAGECLASS_IDX Index

```
Solutions tried 2
Solutions blocks created 1
Created solutions pruned 0
Cost of the chosen solution     4.2839408E+00   !  (4 I/O)
Cardinality of chosen solution  4.5937499E-01   !  (0.5 rows)
Leaf#01 FFirst JOBS Card=15 ❶
  BgrNdx1 WAGECLASS_IDX [1:1] Fan=19 ❷
 JOB_CODE   JOB_TITLE              MINIMUM_SALARY   MAXIMUM_SALARY
 APGM       Associate Programmer         15000.00         24000.00
 DMGR       Department Manager           50000.00        100000.00
 DSUP       Dept. Supervisor             36000.00         60000.00
 EENG       Electrical Engineer          20000.00         40000.00
 MENG       Mechanical Engineer          20000.00         35000.00
 PRGM       Programmer                   20000.00         35000.00
 SANL       Systems Analyst              40000.00         60000.00
 SPGM       Systems Programmer           25000.00         50000.00
8 rows selected
SQL>
```

The display in Example C–11 shows that creating a new index has changed the retrieval strategy selected by the optimizer. Instead of the sequential retrieval strategy found in Example C–9, the availability of the WAGECLASS_IDX produces a dynamic FFirst leaf strategy. The following callouts are keyed to Example C–11:

❶ This line identifies the strategy as the fast first type of dynamic leaf optimization on the JOBS table, which contains 15 rows.

❷ This line shows that the background process opens the WAGECLASS_IDX index.

In this case, the optimizer can locate all rows whose WAGE_CLASS column contains the value 4 by using the sorted index defined for the WAGE_CLASS column. Because the value 4 is repeated for many rows, that is, the column contains duplicate values, all duplicate nodes with key value 4 in the index must be locked and tested. The second column tested is MINIMUM_SALARY. This column is not indexed. By indexing the WAGE_CLASS column, the optimizer must locate all rows that contain the value 4 and, from that group, test each to determine if the row also contains the MINIMUM_SALARY column with a value within the range of values specified in the query.

The optimizer cannot determine from the index alone which rows satisfy the query, because all the information is not in the index. Therefore, the optimizer uses the dbkey from the index to go directly to each row whose WAGE_CLASS column contains the value 4 and test its MINIMUM_SALARY column against the range of values in the second clause, MINIMUM_SALARY BETWEEN 10000 AND 60000.

By using the O flag and S flag together, you can understand why the query optimizer chooses a particular strategy. Notice also that Oracle Rdb uses the value 4 to restrict the search through the index. The display from the SO flags indicates that one low Ikey segment and one high Ikey segment ([1:1]), of the WAGECLASS_IDX sorted index is used to specify the index range to scan in search of this value defined as 4.

The cost associated with this query, using an index defined for the WAGE_CLASS column alone, is estimated at four I/Os. Oracle Rdb retrieves the rows through the index. However, if the rows in a table experience frequent additions and deletions, Oracle Rdb stores them randomly in the logical area associated with the table. Therefore, the access strategies Oracle Rdb chooses to execute this query are different from the strategies it would choose if the table were empty. Defining an index on the WAGE_CLASS column also partially answers a previous question about whether or not an index defined for this table could result in better performance (nine I/Os as opposed to four I/Os when the index is used). You would need to time some queries to determine if the savings in time is actually measurable.

C.6 Displaying the Optimization Strategy and Execution Trace with the SE Flags

During dynamic optimization, the optimizer can change strategies. For this reason, using the S flag to examine optimizer strategy does not reveal all the steps in query execution. When you define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter as SE, Oracle Rdb displays the execution trace for dynamic optimization strategies and includes the following lines:

- One line at the initial stage
- One line at the conclusion or termination of each background index
- One line at the conclusion or termination of the foreground index
- One line at the conclusion or termination of the final phase

In certain situations any of the preceding information can be omitted in the E debug flag display. This indicates that a given phase was not needed for a given leaf run. However, at least one line is always printed.

The "\" debug flag sets the internal dbkey buffer size to a very small (testing) value of 10 dbkeys. This allows you to test dynamic optimization with small tables by forcing the use of temporary tables to store dbkey lists.

Note

The "\" debug flag should be used only for testing or problem solving. It is not intended for production use as it slows down system performance.

Digital UNIX

You can specify the maximum number of leaf node iterations to be printed in the execution trace display by defining the RDB_DEBUG_FLAGS configuration parameter in your .dbsrc configuration file as follows:

```
RDB_DEBUG_FLAGS "SEnnn"
```

Substitute the maximum number of iterations for the notation *nnn*. The default number is 100 iterations. You can set this limitation when you have a complex query where the inner loop of a cross block join strategy contains a leaf node. In some cases, the execution trace for this type of strategy can result in an extremely long display that can slow performance and consume excessive disk space. ♦

The remainder of this section examines detailed output of the dynamic optimization (leaf) strategy using the S and E flags. Table C–3 shows notation used with the E flag display.

Table C–3 Output Definitions for the E Flag

Output	Definition
~E#000*n*.0*n*(*n*)	Identifies an execution trace (E flag) line. For example, the line ~E#0001.01(1) indicates the following: • *#0001* indicates that this is the execution trace for the first query strategy of the session. • *.01* indicates that the output is from the first leaf in the strategy's execution tree. • *(1)* indicates that this is the first iteration of this leaf.
Estim Ndx:Lev/Seps /DBKeys *n:n/n/n n:n /n\n*	Indicates cardinality reestimation of a leaf's background indexes. Refer to Section 5.7.2 for background information. • *Estim* indicates that this is an estimation line. • *Ndx:* indicates the background index number in a sequence delivered by the static optimizer, for example BgrNdx1 or BgrNdx2. • *Lev* indicates a B-tree descent to an index node where a range split occurs or to a level 1 index node. An index node split occurs when that index node no longer covers the entire index key range. In other words, the key range is split across more than one index node. By knowing the level at which the split occurs and the average keys per node, the optimizer can estimate cardinality more accurately than the initial estimate made by the static optimizer. If a level 1 index node is reached, the cardinality is equal to the number of entries in the node that are within the key range times the duplicates key factor. • *Seps* indicates the number of Ikey separators between the range nodes at this level, that is #OfNodes minus 1. • *DBKeys* indicates the number of dbkeys (retrievals) in the range. Each *n* corresponds to a position in the Ndx:Lev/Seps/DBKeys notation. The notation is illustrated in the following example: `~E0005.01(1) Estim  Ndx:Lev/Seps/DBKeys 1:1/1/1 2:3/1\197`

(continued on next page)

Table C–3 (Cont.) Output Definitions for the E Flag

Output	Definition
	• 1:1/1/1 Background index number 1, as indicated in the S display. B-tree descent stopped at level 1, the lowest level. The key range contained one separator. Note that at level 1, Seps is not #OfNodes minus 1 but #OfNodes, that is, the number of entries that could be row dbkeys or dbkeys pointing to the lists of duplicate key dbkeys. One (1) dbkey. The "1" preceded by a slash (/) means the "1" is a precise figure (not an estimate), indicating no duplicate keys are involved. • 2:3/1 \ 197 Background index number 2. Split level 3. Two (2) nodes in the range at this split level (#OfNodes–1=1). 197 estimated dbkeys. When preceded by a backslash (\ ), the number is the *estimated* number of dbkeys (retrievals) in the range. Another example is: `~E0007.01(1) Estim  Ndx:Lev/Seps/DBKeys 2:3/1\38 1:_52` • 2:3/1 \ 38 Background index number 2, as indicated by the S display. Level 3.

(continued on next page)

Table C–3 (Cont.) Output Definitions for the E Flag

Output	Definition
	Two (2) nodes in the range at this split level (#OfNodes–1=1). 38 estimated dbkeys. • 1:_52 Background index number 1. When preceded by an underscore (_), the number is the static optimizer's estimate of the number of records to retrieve. This estimate is provided when the dynamic optimizer estimate cannot be conducted. Background index information is shown in ascending dbkeys order. Therefore, in this example, the background index number 2 information is displayed first because background index number 2 has fewer dbkeys (38) than background index 1 (52).
BgrNdx1	Indicates the first index chosen for the background process. If more than one index is opened, they are numbered consecutively.
FgrNdx	Indicates foreground index. There can be a single foreground index.
DBKeys=*n*	Indicates the number of dbkeys accumulated in a dbkey list (buffer or temporary table) while scanning an index, excluding those dbkeys that are filtered out.
Fetches=*n*+*n*	Indicates the number of the physical I/O operations done to fetch index pages during an index range scan. The left side of the plus sign is the I/O portion for an index open stage (B-tree descent). The right of the plus sign is the get next I/O portion, that is, the index scan or a data scan. The number of I/O operations does *not* accumulate from fetch line to fetch line for a particular leaf scan. Instead, this line shows the number of physical fetches done for a particular background index. Note that this information does not include I/O required for sorts or for moving data to the temporary table.
RecsOut=*n*	Indicates the number of records delivered to a caller by this leaf node at the moment this BgrNdx completes or terminates. The RecsOut count increments with each new ~E line printed for a given leaf scan.
#Bufs=*n*	Indicates the estimated number of table page buffers that will be read when all the rows pointed to by the dbkeys in the dbkey list are fetched. The #Bufs value is displayed for fast first, background only, and index only leaf retrieval strategies.
Fin Buf	Indicates the Fin stage has obtained a dbkey list from a buffer.
Fin Ttbl	Indicates the Fin stage has obtained a dbkey list from a temporary table.

(continued on next page)

Table C–3 (Cont.) Output Definitions for the E Flag

Output	Definition
Fin Seq	Indicates the Fin stage has delivered the rows sequentially. The Fin stage should never deliver rows sequentially, but this can happen if the stored cardinality has become much higher than the actual cardinality.
Fin ?state?	Indicates the Fin stage has not been started because the retrieval was prematurely terminated.
ThreLim	Indicates too many dbkeys have been read. A sequential scan or possibly another index would be faster.
FtchLim	Indicates too many I/O operations have been done. Try the next index or a sequential scan.
Termin*	Indicates background process termination.
EofData	Indicates successful completion of a background index scan.
'CUT	Indicates the background process or foreground process was stopped by an explicit “Close Leaf” command.
'ABA	Indicates a process was abandoned (terminated) in favor of the other process.

Keep the following three rules in mind when you read the output of the S and E flags:

- The ~S (strategy) output lines are printed during the final stage of query compilation. The ~E (execution) output lines are printed during query execution *after* processing has completed; that is, after the Bgr, Fgr, and Fin processes have completed. Note that ~E output lines can appear apart from their ~S output counterparts. They also can be intermixed for different leaves in the same query as well as for the leaves of different queries. This is why the ~S notation contains two numbers and the ~E notation contains three numbers: the numbers identify to which query, to which instance of a leaf within the query, and to which iteration of a given leaf the ~E output lines belong.
- The ~E output lines are printed just *above* the last line of query output (the last delivered row).
- The RecsOut line shows the total count of rows delivered for a run *up to that point*. This means that rows could have been delivered by either the foreground process or the Fin process, but whatever quantity has been delivered by the time the background process finishes is shown on the Bgr line.

The examples in this section show examples of debug flag output with the S and E flags set.

Fast First Leaf Strategy with Execution Trace

Example C–12 illustrates the Fast First (FFirst) leaf retrieval strategy.

Example C–12 SE Flag Display—FFirst Retrieval

```
SQL> SELECT LAST_NAME, EMPLOYEE_ID FROM EMPLOYEES
cont>  WHERE EMPLOYEE_ID > ’00200’;
~S#0004 ❶
Leaf#01 FFirst EMPLOYEES Card=100         ❷
  BgrNdx1 EMP_EMPLOYEE_ID [1:0] Fan=17    ❸
 LAST_NAME        EMPLOYEE_ID
 Clinton          00201                   ❹
 Harrington       00202
 Gaudet           00203
   .
   .
   .
 Dement           00405
 Mistretta        00415
 Ames             00416
 Blount           00418
 MacDonald        00435
~E#0004.01(1) BgrNdx1 EofData  DBKeys=63  Fetches=0+0  RecsOut=63 #Bufs=24 ❺
~E#0004.01(1) FgrNdx  FFirst   DBKeys=63  Fetches=0+23 RecsOut=63‘ABA      ❻
~E#0004.01(1) Fin     Buf      DBKeys=63  Fetches=0+0  RecsOut=63          ❼
 Herbener         00471                   ❽
63 rows selected
SQL>
```

The following callouts are keyed to Example C–12:

❶ This is the fourth query of the session.

❷ This is the first (and only) leaf node in the execution tree for this query. The optimizer chose the fast first leaf type to access the EMPLOYEES table, which has a cardinality of 100 rows.

❸ The background process uses one index (BgrNdx1), EMP_EMPLOYEE_ID, which the optimizer has determined is the best available index. Because this is a FFirst retrieval, both the background process and the foreground process use EMP_EMPLOYEE_ID.

There is one low index key value, but no upper limit for the scan because the query contains the Boolean greater than (>) operator. The fanout factor is an average of 17 branches for each index node.

❹ The foreground process begins delivering rows immediately and delivers 63 rows before the first execution trace line appears. The ~E output lines are always printed one line before the last line of query output (the last delivered row).

❺ This execution trace line contains the following information:

- ~E#0004.01(1)

  This execution trace line is for the fourth query of the current session; for the first leaf in the execution tree; and for the first iteration of this leaf.

- BgrNdx1 EofData DBKeys=63 Fetches=0+0 RecsOut=63 #Bufs=24

  The background process has finished the index scan (EofData). The DBKeys=63 shows that 63 dbkeys were collected in the buffer while scanning the index. The Fetches notation shows that no I/O operations were required for the scan of this leaf. The #Bufs notation is an estimate of the number of table page buffers that will be read when all the rows pointed to by the dbkeys in the dbkey list are fetched.

❻ ~E#0004.01(1) FgrNdx FFirst DBKeys=63 Fetches=0+23 RecsOut=63‘ABA

The FgrNdx FFirst notation indicates that the background index is the source of the dbkeys that the foreground process has delivered. The foreground process itself has not fetched any dbkeys because the RecsOut total of 63 dbkeys has not been incremented above the 63 delivered by the background process. The Fetches notation shows that 23 I/O operations were required for data record fetches. Note that 23 I/O operations is close to the estimate of 24 shown in the #Bufs notation.

The optimizer abandons (‘ABA) the foreground process because the background process has successfully finished. The optimizer is now ready to do the final stage fetches.

❼ ~E#0004.01(1) Fin Buf DBKeys=63 Fetches=0+0 RecsOut=63

This line shows that the final (Fin) stage has read all 63 dbkeys stored in a buffer (Buf), tested them against the dbkey list already delivered by the foreground process, and ignored all 63 dbkeys in the dbkey list as already delivered, thus doing no I/O operations (Fetches=0+0), and, together with the foreground process, has delivered 63 rows (RecsOut=63).

❽ The final output line is always printed after the execution trace line responsible for its delivery.

Sorted Order Leaf Strategy with Execution Trace

Example C–13 illustrates the sorted order leaf retrieval strategy with both the S and E flags set.

Example C–13 SE Flag Display—Sorted Order Retrieval

```
SQL> SELECT EMPLOYEE_ID, LAST_NAME
cont> FROM EMPLOYEES
cont> WHERE LAST_NAME = 'Clarke'
cont> ORDER BY EMPLOYEE_ID;
~S#0004 ❶
Leaf#01 Sorted EMPLOYEES Card=100        ❷
  FgrNdx  EMP_EMPLOYEE_ID [0:0] Fan=17   ❸
  BgrNdx1 EMP_LAST_NAME [1:1] Fan=12     ❹
~E#0004.01(1) BgrNdx1 EofData  DBKeys=3  Fetches=0+0  RecsOut=0 ❺
 EMPLOYEE_ID   LAST_NAME ❻
 00188         Clarke
 00196         Clarke
~E#0004.01(1) FgrNdx  Sorted   DBKeys=5  Fetches=0+4  RecsOut=3 ❼
 00212         Clarke ❽
3 rows selected
SQL>
```

The following callouts are keyed to Example C–13:

❶ This is the fourth query of the session.

❷ This is the first (and only) leaf node in the execution tree for this query. The optimizer chose the sorted order leaf type to access the EMPLOYEES table, which has a cardinality of 100 rows.

❸ The foreground process uses EMP_EMPLOYEE_ID for its index (FgrNdx) scan. This index returns the rows in the specified sorted order. Because the query does not specify a condition on the EMPLOYEE_ID column, the entire index must be scanned as indicated by the [0:0] notation. The fanout factor is 17.

❹ The background process uses one index (BgrNdx1), EMP_LAST_NAME. The background process simultaneously scans the EMP_LAST_NAME index while the foreground process scans EMP_EMPLOYEE_ID. The query specifies an equivalence for the LAST_NAME column (= 'Clarke'); thus the [1:1] notation indicates one low Ikey segment and one high Ikey segment. The fanout factor is 12.

The foreground process does a complete scan over the EMP_EMPLOYEE_ID sorted index. If at any point during the scan no complete background dbkey list is available, the foreground process fetches all the rows for its dbkeys. If at a given point during the scan a complete background dbkey list becomes available, the foreground process fetches only the rows whose dbkeys belong to the background dbkey list.

❺ The background process finishes first, having scanned its index to completion (EofData). The background process found three dbkeys, requiring no I/O operations to get an index page in order to open it and no I/O operations for the index scan. No rows were delivered (RecsOut=0) by the foreground process at this point; instead, the dbkeys were written to a dbkey list.

❻ The foreground process delivers rows.

❼ The foreground process can now use the dbkey list created by the background process and no longer needs to fetch every row from the table. The dbkeys from the foreground index are filtered through the dbkey list; if the dbkey is there, the foreground process fetches the row and matches it against the complete query selection. If the dbkey is not in the dbkey list, the row is not fetched.

The foreground process scanned the entire foreground index (EMP_EMPLOYEE_ID [0:0]) and used five dbkeys (DBKeys=5) to perform data record fetches. It did filter the EMP_EMPLOYEE_ID index dbkeys through the background process' dbkey list and output three rows (RecsOut=3), which required four page I/O operations (Fetches=0+4). The four fetches for the foreground index scan include the index scan itself plus all necessary fetches of data records (using three dbkeys in this case). If the background dbkey list had not been available, fetches for all 100 dbkeys for the entire table would have been necessary.

❽ The final output line is always printed after the execution trace line responsible for its delivery.

Index Only Leaf Strategy with Execution Trace

Example C–14 illustrates the index only leaf retrieval strategy with both the S and E flags set. To illustrate this leaf strategy, two indexes have been created specifically for this example:

```
SQL> CREATE INDEX EMP_LN_FN_SX ON EMPLOYEES
cont> (LAST_NAME, FIRST_NAME, SEX);
SQL>
SQL> CREATE INDEX EMP_SX ON EMPLOYEES
cont> (SEX);
```

The EMP_LN_FN_SX index contains all the columns needed in the sample query.

Example C–14 SE Flag Display—Index Only Leaf Retrieval

```
SQL> SELECT LAST_NAME, FIRST_NAME FROM EMPLOYEES
cont> WHERE FIRST_NAME <> 'Alvin' AND LAST_NAME >= 'L'
cont> AND FIRST_NAME <= 'U' AND SEX = 'F';
~S#0004 ❶
Leaf#01 NdxOnly EMPLOYEES Card=100   ❷
  FgrNdx  EMP_LN_FN_SX [1:0] Fan=9   ❸
  BgrNdx1 EMP_SX [1:1] Fan=19        ❹
 LAST_NAME        FIRST_NAME ❺
 Lapointe         Hope
 Lapointe         Jo Ann
 MacDonald        Johanna
   .
   .
   .
 Ulrich           Christine
 Villari          Christine
 Watters          Christine
~E#0004.01(1) FgrNdx  NdxOnly  DBKeys=16  Fetches=2+0  RecsOut=16 ❻
 Watters          Cora ❼
16 rows selected
SQL>
```

The following callouts are keyed to Example C–14:

❶ This is the fourth query of the session.

❷ This is the first (and only) leaf node in the execution tree for this query. The optimizer chose the index only (NdxOnly) leaf type to access the EMPLOYEES table, which has a cardinality of 100 rows.

❸ The foreground process chose the EMP_LN_FN_SX index (defined specifically for this example) because it contains all the columns needed by the query. There is one low index key value, but no upper limit for the scan. The fanout factor indicates an average of nine branches for each index node.

❹ The background process chose the EMP_SX index (defined specifically for this example), which is based on the SEX column of the EMPLOYEES table. The query specifies an equivalence for the SEX column (= 'F'), thus the [1:1] notation indicates one low index key segment and one high index key segment. The fanout factor is 19.

❺ The foreground process begins returning rows immediately.

❻ The foreground process has completed its scan of the EMP_LN_FN_SX index with two (2) physical I/O operations, found 16 dbkeys, and returned 16 rows. Because the foreground index contained all the columns required by the query, and thus was so efficient, the background index scan was not started.

❼ The final output line is always printed after the execution trace line responsible for its delivery.

Join Using Sorted Order Leaf Strategy with Execution Trace

Example C–15 illustrates a join of two tables that uses the zigzag variation of the match strategy. One loop uses the sorted order leaf retrieval strategy; the second loop uses an index only retrieval strategy. Both the S and E flags are set.

Example C–15 SE Flag Display—Leaf Strategy with Join

```
SQL> SELECT E.EMPLOYEE_ID, JH.JOB_START
cont> FROM EMPLOYEES E, JOB_HISTORY JH
cont> WHERE E.EMPLOYEE_ID = JH.EMPLOYEE_ID
cont> AND (E.LAST_NAME > ′T′ OR E.LAST_NAME > ′Y′ OR E.LAST_NAME > ′Z′);
~S#0005 ❶
Conjunct ❿
Match ❷
  Outer loop ❸
    Leaf#01 Sorted EMPLOYEES Card=100         ❹
      FgrNdx  EMP_EMPLOYEE_ID [0:0] Fan=17    ❺
      BgrNdx1 EMP_LAST_NAME [1:0...]3 Fan=12  ❻
  Inner loop      (zig-zag) ❼
    Get     Retrieval by index of relation JOB_HISTORY  ❽
      Index name  JH_EMPLOYEE_ID [0:0] ❾
 E.EMPLOYEE_ID   JH.JOB_START
 00164            5-Jul-1980  ⓫
~E#0005.01(1) BgrNdx1 EofData  DBKeys=9  Fetches=1+0  RecsOut=1 ⓬
 00164           21-Sep-1981
 00170           26-Nov-1980  ⓭
 00186           25-Apr-1980
 00186            1-Jul-1975
 00186           16-Aug-1977
   .
   .
   .
```

(continued on next page)

Example C–15 (Cont.) SE Flag Display—Leaf Strategy with Join

```
 00242          12-May-1978
 00242          11-May-1980
 00247          19-Jan-1982
 00276           3-Jun-1977
~E#0005.01(1) FgrNdx  Sorted   DBKeys=11  Fetches=0+7  RecsOut=9 ⓮
 00276          15-Mar-1980 ⓯
20 rows selected
SQL>
```

The following callouts are keyed to Example C–15:

❶ This is the fifth query of the session.

❷ The optimizer uses the match strategy (specifically, the zigzag variation) to join the two tables in the query. The two loops of the match strategy access different tables.

❸ The table assigned to the outer loop is scanned in conjunction with the inner loop. Processing in the two loops advances so that the current keys in each loop remain as closely matched as possible. The optimizer, using the key with the lower value, searches for a key in the opposite loop that matches (is equal to) or exceeds (is greater than) the lower valued key.

❹ The optimizer chose the sorted order leaf type to access the EMPLOYEES table, which has a cardinality of 100 rows.

❺ The foreground process uses the EMP_EMPLOYEE_ID index (FgrNdx). This index returns rows in sorted order. The entire index must be scanned as indicated by the [0:0] notation. The fanout factor is 17.

❻ The background process uses one index (BgrNdx1), EMP_LAST_NAME. The background process simultaneously scans the EMP_LAST_NAME index while the foreground process scans EMP_EMPLOYEE_ID. The query specifies 3 ranges for the LAST_NAME column (> ′T′ OR > ′Y′ OR > ′Z′), thus the [1:0 . . .]3 notation indicates one low Ikey segment and no high Ikey segment for three ranges. The fanout factor is 12.

The foreground process does a complete scan over the sorted index EMP_EMPLOYEE_ID. For each dbkey in the index, the foreground fetches a row and then checks to see that the row satisfies the second condition on the EMPLOYEES table, LAST_NAME > ′T′ OR LAST_NAME > ′Y′ OR LAST_NAME > ′Z′. The background process helps by building a dbkey list using the EMP_LAST_NAME index. The dbkey list contains dbkeys of all rows that satisfy the LAST_NAME > ′T′ OR LAST_NAME > ′Y′ OR LAST_NAME > ′Z′ condition. Because the background index scan starts with values greater than 'T', the background has fewer dbkeys to process and finishes before the foreground. The foreground process now filters its dbkeys through the dbkey list and avoids unnecessary row fetches.

Using the sorted order leaf strategy, the outer loop passes one E.EMPLOYEE_ID value at a time to the inner loop.

❼ The table assigned to the inner loop is scanned next. The zigzag strategy is applied to the inner loop.

❽ The optimizer chose indexed access to the JOB_HISTORY table for the inner loop.

❾ The optimizer uses the index JH_EMPLOYEE_ID to access the table JOB_HISTORY. The [0:0] notation indicates the entire index must be scanned. Using the E.EMPLOYEE_ID value from the outer loop, the optimizer searches for all matching values in the JH_EMPLOYEE_ID index.

- If no matching value or values are found, the next highest JH.EMPLOYEE_ID value is returned to the outer loop, where it is compared with the next E.EMPLOYEE_ID value found by the sorted order leaf scan.
- If values match (JH_EMPLOYEE_ID = EMP_EMPLOYEE_ID), the row in the JOB_HISTORY table is fetched (indicated by the Get in the previous line) and the optimizer continues scanning the JH_EMPLOYEE_ID index for any duplicate JH.EMPLOYEE_ID values.

EMPLOYEE_ID values are passed back and forth (zigzag) between loops until no more matches are found.

❿ The Conjunct notation appears above all match strategies. In this example, it is not necessary because the zigzag match strategy has filtered the delivered rows to avoid duplicates. Sometimes, however, if the data types of the columns are different, the match strategy performs a rougher equality check, and the Conjunct is required to filter out possible extra pairs of match keys coming from each loop.

⓫ The foreground process delivers this row before the background process completes its scan.

⓬ The background process finishes first, having scanned its index, EMP_LAST_NAME, to completion (EofData). The background process found nine dbkeys, which required one I/O operation for the index scan. A dbkey list containing nine dbkeys is delivered to the foreground process just as the foreground process starts to scan its index. The foreground process fetches and delivers one row (RecsOut=1). From this point on, the foreground process considers only the nine background dbkeys for fetching rows.

⓭ These rows are delivered by the foreground process.

⓮ The foreground process now uses the dbkey list created by the background process and no longer needs to fetch every row from the table. The dbkeys from the foreground index are filtered through the dbkey list; if the dbkey is there, the foreground process fetches the row and matches it against the complete query selection. If the dbkey is not in the dbkey list, the row is not fetched.

The foreground process filters the EMP_EMPLOYEE_ID index dbkeys through the background process' dbkey list and outputs nine rows (RecsOut=9), which requires seven page I/O operations (Fetches=0+7).

Notice that the nine output rows (RecsOut=9) do not match the actual total output of 20 rows. The inner loop must have output the other rows, and this is confirmed by the results, which show duplicate EMPLOYEE_ID values (00164 and 00186).

⓯ The final output line is always printed after the execution trace line responsible for its delivery.

C.7 Displaying Sort Statistics with the R Flag

OpenVMS VAX OpenVMS Alpha

On OpenVMS, Oracle Rdb displays sort statistics upon completion of a sorting operation when the Oracle Rdb logical name RDMS$DEBUG_FLAGS is defined as R or when the SET FLAGS statement with the SORT_STATISTICS keyword is enabled during query execution. ♦

Digital UNIX

The SORT_STATISTICS keyword and the R debug flag are ignored on Digital UNIX. ♦

OpenVMS VAX OpenVMS Alpha

SQL often performs sorting for the ORDER BY clause and the implicit sorting required for the UNION and GROUP BY clauses. If possible, the optimizer avoids sorting data when an appropriate sorted index is available, or when a sort operation is redundant. In these cases, the optimizer may choose an alternate strategy to retrieve the rows in the correct order without requiring the sorting operation.

Oracle Corporation recommends that the R debug flag always be used in conjunction with the S debug flag (or STRATEGY keyword of the SET FLAGS statement). This allows the sort statistics to be correlated with a known query which is possibly a constraint, trigger, system, or user query. In addition, the SORT_STATISTICS keyword causes the STRATEGY output to contain extra descriptive information.

The sorting operation can consume both CPU and I/O resources. You can use the output of the debug flags to tune the number and location of sort work files, the sort virtual memory, and to help analyze system parameter requirements for some queries.

Example C–16 shows the sort statistics from the WORK_STATUS table in the mf_personnel database.

Example C–16 Strategy Display for Sample Query

```
SQL> SET FLAGS 'STRATEGY';
SQL> SELECT STATUS_CODE FROM WORK_STATUS ORDER BY STATUS_CODE;
Sort
Get     Retrieval sequentially of relation WORK_STATUS
 STATUS_CODE
 0
 1
 2
3 rows selected
```

The SORT statistics are generated only if one or more of the "Sort" keywords appears in the strategy display.

If this query is executed in interactive SQL or dynamic SQL, it is likely that SQL system queries (used to load and validate metadata) might also display strategy and sort statistics. Therefore, it is important to differentiate the SQL queries from the user queries.

Example C–17 shows the use of both the STRATEGY and SORT_STATISTICS keywords.

Example C–17 Sort Statistics for Sample Query

```
SQL> SET FLAGS 'STRATEGY, SORT_STATISTICS';
SQL> SELECT STATUS_CODE FROM WORK_STATUS ORDER BY STATUS_CODE;
Sort
SortId# 4., # Keys 2                                             ❶
  Item# 1, Dtype: 2, Order: 0, Off: 0, Len: 1                    ❷
  Item# 2, Dtype: 14, Order: 0, Off: 1, Len: 1                   ❸
  LRL: 24, NoDups:0, Blks:5, EqlKey:0, WkFls: 2                  ❹
Get     Retrieval sequentially of relation WORK_STATUS
 STATUS_CODE
 0
 1
```

(continued on next page)

Example C–17 (Cont.) Sort Statistics for Sample Query

```
SORT(2) SortId# 4, --------------------- Version: V5-000        ❺
  Records Input: 3     Sorted: 3       Output: 0                 ❻
LogRecLen Input: 24    Intern: 24      Output: 24                ❼
Nodes in SoTree: 112   Init Dispersion Runs: 0                   ❽
Max Merge Order: 0     Numb.of Merge passes: 0
Work File Alloc: 0                                               ❾
MBC for Input: 0       MBC for Output: 0                         ❿
MBF for Input: 0       MBF for Output: 0                         ⓫
Big Allocated Chunk: 88576 idle                                  ⓬
 2
3 rows selected
```

❶ When the STRATEGY keyword is used in conjunction with SORT_ STATISTICS, Oracle Rdb displays extra information about the sort operation. The number of keys (# Keys) indicates that there are two data items: the NULL indicator, and the column being sorted.

❷ **Item# 1** describes the NULL indicator which is of type UNSIGNED BYTE. It will be ordered ASCENDING (0) rather than DESCENDING (1). It occurs at offset 0 in the logical record, and has a length of one byte. Refer to Table C–4 for the data types used by Oracle Rdb when calling the sort interface.

❸ **Item# 2** describes the column STATUS_CODE which is of type CHARACTER; it will be ordered ASCENDING (0). It occurs at offset 1 in the logical record, and has a length of one byte. Refer to Table C–4 for the data types used by Oracle Rdb when calling the sort interface.

❹ **LRL** (logical record length): in this case is 24 bytes. Oracle Rdb allocates space for the NULL indicator, columns from the table, and room for two dbkey fields. The dbkeys are aligned on a quadword boundary. If the data is derived from other sources (such as a UNION), then dbkey access is not possible and these extra dbkey fields are not part of the logical record.

NoDups: When this value is zero, then the sort allows duplicates. If the value is one, then duplicates are not allowed. See Example C–18 and Example C–20, which show how both DISTINCT and UNION set this attribute to one to force the sort operation to discard duplicate rows.

Blks: This is the estimated size of the data to be sorted. It is based on the estimated cardinality for the query (displayed by RDMS$DEBUG_FLAGS O, or the SET FLAGS ESTIMATES option), and the logical record length (LRL).

EqlKey: This flag indicates if a specialized equals callback routine is provided to the sort interface. This flag is currently always zero, indicating that no special routine is used.

WkFls: This value reflects the number of sort work files established with the RDMS$BIND_SORT_WORKFILES logical name on OpenVMS.

❺ **SortId#** shows that this sort operation is related to the sort operation used in the strategy display. The **Version** is the internal Oracle Rdb version of the sort interface.

❻ **Records Input** describes the number of rows passed to the sort interface. **Sorted** describes the number of rows actually sorted. The file interface to the sort operation allows the user to omit rows during input; however, this value will be identical to the **Records Input** value for Oracle Rdb because it uses the record interface. **Output** will always be zero for Oracle Rdb.

❼ **LogRecLen Input** is the logical record length. This can be calculated based on the column values and NULL indicators used by the query. Note that the sort operation may be sorting the results of a join or computation. **Intern** and **Output** will always be the same as **LogRecLen Input** because Oracle Rdb only uses a record sort algorithm.

❽ **Nodes in SoTree**, **Init Dispersion Runs**, **Max Merge Order**, and **Numb.of Merge passes** describe the sort operations.

❾ **Work File Alloc** indicates how many work files were used in the sort operation. A value of zero indicates that the sort was accomplished completely in memory.

❿ **MBC for Input** and **MBC for Output** is the OpenVMS RMS multiblock count for the input files. It is always zero, because Oracle Rdb uses the sort record interface, not the file interface.

⓫ **MBF for Input** and **MBF for Output** is the OpenVMS RMS multibuffer count for the input files. It is always zero, because Oracle Rdb uses the sort record interface, not the file interface.

⓬ **Big Allocated Chunk** is the amount of virtual memory that has been allocated to the sorting function. Oracle Rdb retains this memory for future sort requests, and may grow it if it is too small. By retaining the allocated memory, Oracle Rdb avoids excessive calls to the OpenVMS memory system services to allocate and free virtual memory.

"Idle" indicates that this block of memory was not needed for this sort operation, and "In Use" indicates that the currently described sort operation made use of some or all of this allocated virtual memory.

The decision of whether or not Oracle Rdb uses the **Big Allocated Chunk** and how many bytes to allocate is based on the estimated cardinality of the query solution (obtained by the RDMS$DEBUG_FLAGS O, or the SET FLAGS ESTIMATES option). Sort is initialized with this estimated cardinality before any records are retrieved from the database. In certain cases, this may cause a large number of page faults when the sort operation sets up its sort tree in preparation for the actual sorting. As the solution cardinality is only a mathematical estimate of the number of rows to be retrieved, it may differ from the actual number of rows returned.

Example C–18 shows the change in output when a DISTINCT clause is used to reduce the output to a set of distinct rows.

Example C–18 Effect of DISTINCT Clause on Sort Attributes

```
SQL> SELECT DISTINCT STATUS_CODE
cont> FROM WORK_STATUS WHERE STATUS_CODE >= '1' ORDER BY 1;
Reduce  Sort
SortId# 8., # Keys 2
  Item# 1, Dtype: 2, Order: 0, Off: 0, Len: 1
  Item# 2, Dtype: 14, Order: 0, Off: 1, Len: 1
  LRL: 24, NoDups:1, Blks:5, EqlKey:0, WkFls: 2   ❶
Conjunct        Get     Retrieval sequentially of relation WORK_STATUS
 STATUS_CODE
 1

SORT(6) SortId# 8, --------------------- Version: V5-000
  Records Input: 2      Sorted: 2       Output: 0
LogRecLen Input: 24     Intern: 24      Output: 24
Nodes in SoTree: 112    Init Dispersion Runs: 0
Max Merge Order: 0      Numb.of Merge passes: 0
Work File Alloc: 0
MBC for Input: 0        MBC for Output: 0
MBF for Input: 0        MBF for Output: 0
Big Allocated Chunk: 88576 idle
 2
2 rows selected
```

❶ **NoDups** is set to one to indicate that rows with duplicate values of the sort key should be discarded. This change is made because of the DISTINCT clause in the query (indicated by the Reduce keyword in the strategy display).

Example C–19 shows the change in output when no sort is needed. The optimizer generates the access strategy based on estimates and so always generates the sort operation, even when zero or one row are returned. In such cases, the sort operation can be avoided, with a small saving of CPU time.

Example C–19 Avoided Sort Operation

```
SQL> SELECT STATUS_CODE
cont> FROM WORK_STATUS
cont> WHERE STATUS_CODE >= ’4’ ORDER BY 1;
Sort
SortId# 5., # Keys 2
  Item# 1, Dtype: 2, Order: 0, Off: 0, Len: 1
  Item# 2, Dtype: 14, Order: 0, Off: 1, Len: 1
  LRL: 24, NoDups:0, Blks:5, EqlKey:0, WkFls: 2
Conjunct        Get     Retrieval sequentially of relation WORK_STATUS

SORT(3), SortId# 5 ---- Avoided                                 ❶
 No records
0 rows selected

SQL> SELECT STATUS_CODE
cont> FROM WORK_STATUS
cont> WHERE STATUS_CODE >= ’2’ ORDER BY 1;
Sort
SortId# 6., # Keys 2
  Item# 1, Dtype: 2, Order: 0, Off: 0, Len: 1
  Item# 2, Dtype: 14, Order: 0, Off: 1, Len: 1
  LRL: 24, NoDups:0, Blks:5, EqlKey:0, WkFls: 2
Conjunct        Get     Retrieval sequentially of relation WORK_STATUS

SORT(4), SortId# 6 ---- Avoided                                 ❶
 No records
 STATUS_CODE
 2
1 row selected
```

❶ When the input stream contains one or zero rows, the entire sort operation can be avoided.

Example C–20 shows an example of a UNION generating sort calls.

Example C–20 Query Using UNION Generating SORT

```
SQL> SELECT LAST_NAME FROM EMPLOYEES
cont> UNION
cont> SELECT LAST_NAME FROM CANDIDATES
cont> LIMIT TO 10 ROWS;
Firstn  Reduce  Sort                                             ❶
SortId# 3., # Keys 2
  Item# 1, Dtype: 2, Order: 0, Off: 0, Len: 1
  Item# 2, Dtype: 14, Order: 0, Off: 1, Len: 14
  LRL: 15, NoDups:1, Blks:6, EqlKey:0, WkFls: 2
Merge of 2 entries
  Merge block entry 1
  Get     Retrieval sequentially of relation EMPLOYEES
  Merge block entry 2
  Get     Retrieval sequentially of relation CANDIDATES
 LAST_NAME
 Ames
 Andriola
 Babbin
 Bartlett
 Belliveau
 Blount
 Boswick
 Boyd
 Brown

SORT(1) SortId# 3, --------------------- Version: V5-000
  Records Input: 103    Sorted: 103     Output: 0                ❷
LogRecLen Input: 15     Intern: 15      Output: 15
Nodes in SoTree: 212    Init Dispersion Runs: 0
Max Merge Order: 0      Numb.of Merge passes: 0
Work File Alloc: 0
MBC for Input: 0        MBC for Output: 0
MBF for Input: 0        MBF for Output: 0
Big Allocated Chunk: 88576 idle
 Burton
10 rows selected                                                 ❸
```

❶ UNION is defined as returning a unique set of rows. Therefore, in this example, the merged set of rows are sorted (the **Sort** keyword), then reduced to a distinct set of rows (the **Reduce** keyword), and finally delivered to the application.

The UNION ALL clause returns all rows from the two selection statements and will not normally include a sort operation.

❷ This shows that the exact number of records passed to the sort operation was 103 rows.

❸ The output from interactive SQL shows that only 10 rows were returned. The LIMIT TO clause was used to limit the amount of output for this example. The LIMIT TO (see the Firstn keyword in ❶) is always performed last, after all sorting is completed.

Table C–4 shows the data types used with the sort interface.

Table C–4 Data Types Used with Sort Interface

Type Name	Value	SQL Data Types
DSC$K_DTYPE_ADT	35	DATE (VMS and ANSI), TIME, TIMESTAMP
DSC$K_DTYPE_B	6	TINYINT
DSC$K_DTYPE_BU	2	used for the NULL indicator
DSC$K_DTYPE_F	10	REAL
DSC$K_DTYPE_G	27	DOUBLE PRECISION
DSC$K_DTYPE_L	8	INTEGER
DSC$K_DTYPE_Q	9	BIGINT, INTERVAL
DSC$K_DTYPE_T	14	CHARACTER
DSC$K_DTYPE_VT	37	CHARACTER VARYING (VARCHAR)
DSC$K_DTYPE_W	7	SMALLINT

♦

C.8 Displaying Transaction Activity with the T Flag

When you define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter as T, Oracle Rdb dumps the transaction parameter block generated by the SET (or DECLARE) TRANSACTION statement and also displays transactions when they are committed or rolled back. This information is valuable when tracing the transaction activity of an application. Example C–21 shows a sample display.

Example C–21 Displaying Transaction Activity with the Transaction (T) Flag

```
SQL> SET TRANSACTION
cont>     READ WRITE
cont>     WAIT 30
```

(continued on next page)

Example C–21 (Cont.) Displaying Transaction Activity with the Transaction (T) Flag

```
cont>     RESERVING EMPLOYEES FOR PROTECTED READ,
cont>          DEPARTMENTS FOR EXCLUSIVE WRITE,
cont>          WORK_STATUS FOR SHARED READ
cont>     ISOLATION LEVEL SERIALIZABLE
cont>     EVALUATING EMPLOYEES_PRIMARY_EMPLOYEE_ID AT COMMIT TIME;
 Compile transaction on db: X00000001
~T Transaction Parameter Block: (len=6)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_ISOLATION_LEVEL3 (serializable)
0002 (00002) TPB$K_WAIT_INTERVAL 30 seconds
0005 (00005) TPB$K_WRITE (read write)
 Start_transaction on db: X00000001
 Commit_transaction on db: X00000001
 Prepare_transaction on db: X00000001
~T Transaction Parameter Block: (len=77)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_ISOLATION_LEVEL3 (serializable)
0002 (00002) TPB$K_WAIT_INTERVAL 30 seconds
0005 (00005) TPB$K_WRITE (read write)
0006 (00006) TPB$K_COMMIT_TIME (evaluating) "EMPLOYEES_PRIMARY_EMPLOYEE_ID"
0025 (00037) TPB$K_LOCK_READ (reserving) "EMPLOYEES" TPB$K_PROTECTED
0031 (00049) TPB$K_LOCK_WRITE (reserving) "DEPARTMENTS" TPB$K_EXCLUSIVE
003F (00063) TPB$K_LOCK_READ (reserving) "WORK_STATUS" TPB$K_SHARED
SQL> ROLLBACK;
 Rollback_transaction on db: X00000001
SQL>
```

The start, prepare, commit, and rollback of transactions also indicate the database handle upon which the transaction is operating.

Table C–5 shows the transaction parameter block code that is generated when the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is set to “T”, and provides a brief description of each code.

Table C–5 Transaction Parameter Block (TPB) Information

TPB Code	Description
TPB$K_WAIT	Transaction waits for locks indefinitely; deadlocks are reported.

(continued on next page)

Table C–5 (Cont.) Transaction Parameter Block (TPB) Information

TPB Code	Description
TPB$K_NOWAIT	Transaction does not wait for locks, but reports lock conflicts.
TPB$K_WAIT_INTERVAL	WAIT was specified with timeout interval, so transaction waits for the specified number of seconds.
TPB$K_READ	A read-only transaction.
TPB$K_WRITE	A read/write transaction.
TPB$K_BATCH_UPDATE	A batch-update transaction.
TPB$K_LOCK_READ	The named table is reserved for READ. The modes can be SHARED, PROTECTED, or EXCLUSIVE.
TPB$K_LOCK_WRITE	The named table is reserved for WRITE. The modes can be SHARED, PROTECTED, or EXCLUSIVE.
TPB$K_VERB_TIME	The named constraint has its evaluation time changed to VERB TIME (NOT DEFERRABLE).
TPB$K_COMMIT_TIME	The named constraint has its evaluation time changed to COMMIT TIME (DEFERRABLE).
TPB$K_ISOLATION_LEVEL1	Isolation level read committed.
TPB$K_ISOLATION_LEVEL2	Isolation level repeatable read.
TPB$K_ISOLATION_LEVEL3	Isolation level serializable.
TPB$K_DEGREE3	CONSISTENCY LEVEL 3 (serializable) is deprecated; use ISOLATION LEVEL SERIALIZABLE instead.
TPB$K_DEGREE2	CONSISTENCY LEVEL 2 (read committed) is deprecated; use ISOLATION LEVEL READ COMMITTED instead.

When you define RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS as T, the display indicates when read-only transactions are upgraded to read/write transactions because the database has snapshot files disabled, as shown in Example C–22.

Example C–22 Displaying Read-Only Transactions Upgraded to Read/Write Transactions When Snapshot Files Are Disabled

```
SQL> ALTER DATABASE FILENAME mf_personnel SNAPSHOT DISABLED;
SQL> --
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION READ ONLY;
 Compile transaction on db: X00000002
~T Transaction Parameter Block: (len=2)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_READ (read only)
 Start_transaction on db: X00000002
~T Snapshots are disabled, READ ONLY converted to READ WRITE
SQL>
```

C.9 Logging the TRACE Control Statement with the Xt Flag

The SQL TRACE control statement writes values to a log file after the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter is defined as Xt. The TRACE control statement lets you specify multiple value expressions. It stores a value in a log file for each value expression it evaluates.

SQL turns on trace logging only if the logical name RDMS$DEBUG_FLAGS or the configuration parameter RDB_DEBUG_FLAGS is defined to be Xt. The letter X must be an uppercase letter and the letter t must be in lowercase. The TRACE control statement has no effect when RDMS$DEBUG_FLAGS or RDB_DEBUG_FLAGS is not defined as Xt.

Trace logging can help you debug complex multistatement procedures.

See the *Oracle Rdb7 SQL Reference Manual* for more information on the TRACE control statement.

Index