Oracle® Rdb for OpenVMS

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Oracle® Rdb for OpenVMS

Release Notes

Release 7.1.4

January 2005

Oracle Rdb Release Notes, Release 7.1.4 for OpenVMS

Contents

Preface

Purpose of This Manual

This manual contains release notes for Oracle Rdb Release 7.1.4. The notes describe changed and enhanced features; upgrade and compatibility information; new and existing software problems and restrictions; and software and documentation corrections.

Intended Audience

This manual is intended for use by all Oracle Rdb users. Read this manual before you install, upgrade, or use Oracle Rdb Release 7.1.4.

Document Structure

This manual consists of the following chapters:

Chapter 1	Describes how to install Oracle Rdb Release 7.1.4.
Chapter 2	Describes software errors corrected in Oracle Rdb Release 7.1.4.
Chapter 3	Describes software errors corrected in Oracle Rdb Release 7.1.3.
Chapter 4	Describes enhancements introduced in Oracle Rdb Release 7.1.4.
Chapter 5	Describes enhancements introduced in Oracle Rdb Release 7.1.3.
Chapter 6	Provides information not currently available in the Oracle Rdb documentation set.
Chapter 7	Describes problems, restrictions, and workarounds known to exist in Oracle Rdb Release 7.1.4.

Chapter 1
Installing Oracle Rdb Release 7.1.4

This software update is installed using the standard OpenVMS Install Utility.

NOTE

All Oracle Rdb Release 7.1 kits are full kits. There is no requirement to install any prior release of Oracle Rdb when installing new Rdb Release 7.1 kits.

1.1 Alpha EV7 Processor Support

For this release of Oracle Rdb, the Alpha EV7 (also known as the Alpha 21364) processor is the newest processor supported.

1.2 Oracle Rdb V7.1 Version Numbering Enhancement

Previously, the Oracle Rdb version number was specified as 4 digits (for example, version "7.1.0.2"). Starting with Oracle Rdb Release 7.1.1, an additional, fifth, digit has been added to the kit version number. This new digit is intended to indicate an optimization level of the Rdb software. The use of this new digit is to indicate a "generic" kit (final digit of zero) for all Alpha processors or a "performance" kit that will run on a subset of the supported platforms (final digit of 1). In the future, additional values may be specified to indicate other performance or platform options.

For Oracle Rdb Release 7.1.4, the two kits are 7.1.4.0.0 (compiled for all Alpha processor types) and 7.1.4.0.1 (compiled for EV56 and later Alpha processors). These kits offer identical functionality and differ only in a potential performance difference.

1.3 Requirements

The following conditions must be met in order to install this software:

- Oracle Rdb must be shutdown before you install this update kit. That is, the command file SYS$STARTUP:RMONSTOP71.COM should be executed before proceeding with this installation. If you have an OpenVMS cluster, you must shutdown the Rdb 7.1 monitor on all nodes in the cluster before proceeding.
- The installation requires approximately 280,000 blocks for OpenVMS Alpha systems.
- If you are running Hot Standby and you are upgrading from a version of Oracle Rdb 7.1 prior to 7.1.1, you must install this kit on both the master and the standby systems prior to restarting Hot Standby. This requirement is necessary due to changes to the message format used to transmit journal state information from the master to the standby system.

1.4 Invoking VMSINSTAL

To start the installation procedure, invoke the VMSINSTAL command procedure as in the following examples.

To install the Oracle Rdb for OpenVMS Alpha kit that is compiled to run on all Alpha platforms:

```
@SYS$UPDATE:VMSINSTAL RDBV71400AM device−name OPTIONS N
```

To install the Oracle Rdb for OpenVMS Alpha kit that is performance targeted for Alpha EV56 and later platforms:

```
@SYS$UPDATE:VMSINSTAL RDBV71401AM device−name OPTIONS N
```

device−name

Use the name of the device on which the media is mounted. If the device is a disk drive, you also need to specify a directory. For example: *DKA400:[RDB.KIT]*

OPTIONS N

This parameter prints the release notes.

The full Oracle Rdb Release 7.1 Installation Guide is also available on MetaLink in Adobe Acrobat PDF format:

```
Top Tech Docs\Oracle Rdb\Documentation\Rdb 7.1 Installation and Configuration
Guide
```

1.5 Stopping the Installation

To stop the installation procedure at any time, press Ctrl/Y. When you press Ctrl/Y, the installation procedure deletes all files it has created up to that point and exits. You can then start the installation again.

If VMSINSTAL detects any problems during the installation, it notifies you and a prompt asks if you want to continue. You might want to continue the installation to see if any additional problems occur. However, the copy of Oracle Rdb installed will probably not be usable.

1.6 After Installing Oracle Rdb

This update provides a new Oracle Rdb Oracle TRACE facility definition. Any Oracle TRACE selections that reference Oracle Rdb will need to be redefined to reflect the new facility version number for the updated Oracle Rdb facility definition, "RDBVMSV7.1−4".

If you have Oracle TRACE installed on your system and you would like to collect for Oracle Rdb, you must insert the new Oracle Rdb facility definition included with this update kit.

The installation procedure inserts the Oracle Rdb facility definition into a library file called EPC$FACILITY.TLB. To be able to collect Oracle Rdb event−data using Oracle TRACE, you must move this facility definition into the Oracle TRACE administration database. Perform the following steps:

1. Extract the definition from the facility library to a file (in this case, RDBVMS.EPC$DEF).

   ```
   $ LIBRARY /TEXT /EXTRACT=RDBVMSV7.1−4 −
   _$ /OUT=RDBVMS.EPC$DEF SYS$SHARE:EPC$FACILITY.TLB
   ```
2. Insert the facility definition into the Oracle TRACE administration database.

   ```
   $ COLLECT INSERT DEFINITION RDBVMS.EPC$DEF /REPLACE
   ```

Note that the process executing the INSERT DEFINITION command must use the version of Oracle Rdb that matches the version used to create the Oracle TRACE administration database or the INSERT DEFINITION command will fail.

1.7 Patches for OpenVMS V7.3−1

Several problems that affect installations using Oracle Rdb on OpenVMS V7.3−1 are corrected in patch kits available from HP OpenVMS support. Oracle recommends that you consult with Hewlett−Packard and install these patch kits (or their replacements) to correct or avoid the following problems:

- VMS731_SYS−V0400 corrects the following problems seen with Oracle Rdb:
    - ♦ When using Oracle Rdb Galaxy support, or memory−resident global sections, processes enter a permanent RWAST state at image exit. The system must be rebooted to remove the process and continue normal operations. Note that when using Oracle Rdb Release 7.1.2 databases with SHARED MEMORY IS PROCESS RESIDENT attribute, the Row Cache feature and caches with the SHARED MEMORY IS SYSTEM, LARGE MEMORY IS ENABLED, or RESIDENT attributes, or in an OpenVMS Galaxy configuration with Oracle Rdb Galaxy support enabled, you are at an elevated risk of experiencing this problem.
      Configurations that do not have this patch, or it's future replacement, applied will not be supported by Oracle if the SHARED MEMORY IS PROCESS RESIDENT, the Row Cache, or Galaxy support features are in use. If you are not using these features, then the patch or it's replacement is not mandatory. However, Oracle still strongly recommends that it be used.
    - ♦ Applications using the Oracle Rdb Row Cache or AIJ Log Server (ALS) features would sometimes have their server processes hang in HIB (hibernate) state.
- VMS731_SYSLOA−V0100 corrects the following problem seen with Oracle Rdb:
    - ♦ In an OpenVMS cluster environment, unreported deadlocks and hangs can occur. This problem is sometimes characterized by an Oracle Rdb blocking lock incorrectly being shown as owned by the system (in other words, with a zero PID).

1.8 Oracle Rdb Release 7.1.4.0.1 Optimized for Alpha EV56 (21164A Processor Chip) and Later Platforms

Oracle will be releasing Oracle Rdb 7.1 and later kits in parallel build streams − a "generic" kit that will run on all certified and supported Alpha platforms as well as a "performance" kit that will run on a subset of the supported platforms. The performance kit is intended for those customers with "newer" Alpha processor chips who need higher levels of performance than are offered by the generic kits. The performance kits are otherwise functionally identical to the generic kits.

Oracle will continue to release both types of kits for Oracle Rdb Release 7.1 as long as there is significant customer interest in the generic kit.

For improved performance on current generation Alpha processors, Oracle Rdb Release 7.1.4.0.1 is compiled explicitly for Alpha EV56 and later systems. This version of Oracle Rdb requires a system with a minimum Alpha processor chip of EV56 and a maximum processor chip of Alpha EV7 (known as the Alpha 21364).

Oracle Rdb Release 7.1.4.0.1 is functionally equivalent to Oracle Rdb Release 7.1.4.0.0 and was built from the same source code. The only difference is a potentially improved level of performance. Oracle Rdb Releases 7.1.4.0.0 and 7.1.4.0.1 are certified on all supported Alpha processor types (up to and including the Alpha EV7 processor).

In Release 7.1.4.0.1, Oracle Rdb is explicitly compiled for EV56 and later Alpha processors such that the generated instruction stream can utilize the byte/word extension (BWX) of the Alpha architecture. Additionally, this kit is compiled with instruction tuning biased for performance of Alpha EV6 and later systems that support quad−issue instruction scheduling.

Note that you should not install Release 7.1.4.0.1 of Oracle Rdb on Alpha EV4, EV45 or EV5 systems. These processor types do not support the required byte/word extension (BWX) of the Alpha architecture. Also ensure that all systems in a cluster sharing the system disk are using a minimum of the Alpha EV56 processor.

To easily determine the processor type of a running OpenVMS Alpha system, use the CLUE CONFIG command of the OpenVMS System Dump Analyzer utility (accessed with the ANALYZE/SYSTEM command). The "CPU TYPE" field indicates the processor type as demonstrated in the following example from an HP AlphaServer GS140 6/525 system with an EV6 (21264) processor:

```
$ ANALYZE/SYSTEM
SDA> CLUE CONFIG
System Configuration:
    .
    .
    .
Per−CPU Slot Processor Information:
CPU ID    00                  CPU State    rc,pa,pp,cv,pv,pmv,pl
CPU Type  EV6  Pass 2.3 (21264)
PAL Code  1.96−1              Halt PC      00000000.20000000
    .
    .
    .
```

1.8.1 AlphaServer 4000 EV56 299Mhz Not Supported by Oracle Rdb Release Optimized for Alpha EV56 Processor

Oracle Rdb releases that are optimized for the Alpha EV56 and later processors are not able to run on the AlphaServer 4000 with the 299Mhz EV56 processor. Though this CPU claims to be an EV56, it does not, in fact, implement the byte/word instruction set as required.

According to information on the HP web site, this problem may be present in the AlphaServer 4000 or 4100 systems with a processor module of KN304−FA or KN304−FB. The systems effected appear to include the AlphaServer 4x00 5/300 pedestal, cabinet and rackmount systems: DA−51FAB−ED/−FD/−GB or DA−53GEB−CA/−EA/−FA/−GA.

The indicated CPU is not able to run Oracle Rdb releases that are optimized for the Alpha EV56 and later processors. This effects Oracle Rdb Releases optimized for the Alpha EV56 and later processors.

Possible workarounds include updating the system to an EV56 module for the AlphaServer 4x00 that is later than the KN304−FA or FB (ie a clock speed greater than 300Mhz). Some of the possible modules would include: KN304−AA 400mhz, KN304−DA 466mhz, B3005−CA 533mhz, B3006−EB 600mhz.

Otherwise, an Oracle Rdb release that is not optimized for the Alpha EV56 and later processors must be used (such as Oracle Rdb Releases 7.1.4.0.0)

Please contact your HP AlphaServer hardware vendor for additional information.

1.9 Maximum OpenVMS Version Check Added

As of Oracle Rdb7 Release 7.0.1.5, a maximum OpenVMS version check has been added to the product. Oracle Rdb has always had a minimum OpenVMS version requirement. With 7.0.1.5 and for all future Oracle Rdb releases, we have expanded this concept to include a maximum VMS version check and a maximum supported processor hardware check. The reason for this check is to improve product quality.

OpenVMS Version 8.2−x is the maximum supported version of OpenVMS.

The check for the OpenVMS operating system version and supported hardware platforms is performed both at installation time and at runtime. If either a non−certified version of OpenVMS or hardware platform is detected during installation, the installation will abort. If a non−certified version of OpenVMS or hardware platform is detected at runtime, Oracle Rdb will not start.

1.10 VMS$MEM_RESIDENT_USER Rights Identifier Required

Oracle Rdb Version 7.1 introduced additional privilege enforcement for the database or row cache attributes RESIDENT, SHARED MEMORY IS SYSTEM and LARGE MEMORY IS ENABLED. If a database utilizes any of these features, then the user account that opens the database must be granted the VMS$MEM_RESIDENT_USER rights identifier.

Oracle recommends that the RMU/OPEN command be used when utilizing these features.

Chapter 2
Software Errors Fixed in Oracle Rdb Release 7.1.4

This chapter describes software errors that are fixed by Oracle Rdb Release 7.1.4.

2.1 Software Errors Fixed That Apply to All Interfaces

2.1.1 Redundant Index in Dynamic Index Only Tactic

Bug 3439578

Where multiple indexes appeared useful for retrieving data for a query, and at least one of the indexes contained all columns referenced by the query, the dynamic optimizer might scan additional indexes even though the conditions used had been used on a previous index lookup. This resulted in wasted I/O and poor performance.

In the following example, the columns used for both index lookups on table TTOS270 are identical, but simply in a different order.

```
~S#0003
Tables:
  0 = TTOS275
  1 = TTOS270
Firstn: 100000
Cross block of 2 entries
  Cross block entry 1
    Conjunct: (0.VALIDITY_DATE_FROM < DATE '2001−08−09') AND (SUBSTRING (
              0.GROUP_CODE FROM (1 − 1) FOR 2) = 'SG')
    Index only retrieval of relation 0:TTOS275
      Index name  P_TTOS275 [0:0]
  Cross block entry 2
    Conjunct: <agg0> = 0
    Aggregate−F1: 0:COUNT−ANY (<subselect>)
    Leaf#01 NdxOnly 1:TTOS270 Card=1290000
      Bool: (0.ACCOMM_CODE = 1.ACCOMM_CODE) AND (0.UNIT_TYPE_CODE =
            1.UNIT_TYPE_CODE) AND (0.GROUP_CODE = 1.GROUP_CODE) AND (
            0.ALLOCATION_PRODUCT = 1.ALLOCATION_PRODUCT) AND (
            0.VALIDITY_DATE_FROM >= 1.VALIDITY_DATE_FROM) AND (
            0.VALIDITY_DATE_FROM <= 1.VALIDITY_DATE_TO)
      FgrNdx  P_TTOS270 [4:5] Fan=15
        Keys: (0.GROUP_CODE = 1.GROUP_CODE) AND (0.ALLOCATION_PRODUCT =
              1.ALLOCATION_PRODUCT) AND (0.ACCOMM_CODE = 1.ACCOMM_CODE) AND (
              0.UNIT_TYPE_CODE = 1.UNIT_TYPE_CODE) AND (0.VALIDITY_DATE_FROM >=
              1.VALIDITY_DATE_FROM)
      BgrNdx1 S_TTOS270 [4:5] Fan=15
        Keys: (0.ACCOMM_CODE = 1.ACCOMM_CODE) AND (0.UNIT_TYPE_CODE =
              1.UNIT_TYPE_CODE) AND (0.GROUP_CODE = 1.GROUP_CODE) AND (
              0.ALLOCATION_PRODUCT = 1.ALLOCATION_PRODUCT) AND (
              0.VALIDITY_DATE_FROM >= 1.VALIDITY_DATE_FROM)
```

Oracle Rdb now detects this and does not use the redundant index. This results in a static index only tactic with one index, as shown in the following example.

```
~S#0003
Tables:
  0 = TTOS275
  1 = TTOS270
Firstn: 100000
Cross block of 2 entries
  Cross block entry 1
```

```
    Conjunct: (0.VALIDITY_DATE_FROM < DATE '2001-08-09') AND (SUBSTRING (
              0.GROUP_CODE FROM (1 - 1) FOR 2) = 'SG')
    Index only retrieval of relation 0:TTOS275
      Index name  P_TTOS275 [0:0]
  Cross block entry 2
    Conjunct: <agg0> = 0
    Aggregate-F1: 0:COUNT-ANY (<subselect>)
    Conjunct: 0.VALIDITY_DATE_FROM <= 1.VALIDITY_DATE_TO
    Index only retrieval of relation 1:TTOS270
      Index name  P_TTOS270 [4:5]
        Keys: (0.GROUP_CODE = 1.GROUP_CODE) AND (0.ALLOCATION_PRODUCT =
              1.ALLOCATION_PRODUCT) AND (0.ACCOMM_CODE = 1.ACCOMM_CODE) AND (
              0.UNIT_TYPE_CODE = 1.UNIT_TYPE_CODE) AND (0.VALIDITY_DATE_FROM >=
              1.VALIDITY_DATE_FROM)
```

The problem can be avoided by using the RDMS$MAX_STABILITY logical name to disable the dynamic optimizer.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.2 Corruption After Tables Dropped/Added

Bug 3847689

It was possible for a database to become corrupt after tables were dropped and then added if the Row Cache or PAGE TRANSFER VIA MEMORY features were enabled. Also, database recovery (DBR) processes could fail with a bugcheck similar to the following:

```
***** Exception at 00071050 : DBR$DO_C_AIJBUF + 00000650
%COSI-F-BUGCHECK, internal consistency failure
```

or

```
***** Exception at 0007C538 : DBR$RECOVER_ALL + 00000858
%COSI-F-BUGCHECK, internal consistency failure
```

Verification of the affected database could show various errors, such as:

```
%RMU-W-INVRELID, invalid relation id at dbkey 57:5:0
                  expected relation id 32, found 30
```

or

```
%RMU-W-CANTFINDLAREA, cannot locate logical area 312 in area inventory page list
%RMU-E-BDLAREADY, error readying logical area with dbid 312
```

RMU/VERIFY could also fail with an error similar to the following:

```
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual address=
000000000000000B, PC=00000000003B4AD8, PS=0000001B
%RMU-F-FATALOSI, Fatal error from the Operating System Interface.
%RMU-F-FTL_VER, Fatal error for VERIFY operation at 19-JUL-2004 06:36:25.24
%RMU-F-FTL_VER, Fatal error for VERIFY operation at 19-JUL-2004 06:36:25.24
```

This problem would occur after a table was dropped, then another table was created, and then a process failed that required the DBR process to reapply changes made to the database prior to the time that the table was dropped. When the PAGE TRANSFER VIA MEMORY feature was enabled and the terminated database user's last checkpoint was done prior to the DROP TABLE, then this problem could be encountered.

When Row Cache was enabled, and the oldest checkpoint in the database was done before the DROP TABLE, and the Row Cache Server (RCS) was terminated abnormally (for example, due to a system failure, RCS process failure, etc.) then this problem could be encountered.

The following script demonstrates the problem:

```
$ ! Create a database with the PAGE TRANSFER VIA MEMORY
$ ! feature enabled.
$
$ SQL$
CREATE DATABASE FILENAME TEST
  NUMBER OF CLUSTER NODES 1
  GLOBAL BUFFERS ARE ENABLED (PAGE TRANSFER VIA MEMORY)
  CREATE STORAGE AREA RDB$SYSTEM FILENAME TEST
  CREATE STORAGE AREA TEST_DATA FILENAME TEST_DATA;
DISCONNECT ALL;

ALTER DATABASE FILE TEST
  JOURNAL IS ENABLED
    (FAST COMMIT IS ENABLED)
    ADD JOURNAL AIJ_1 FILENAME TEST;

ATTACH 'FILE TEST';
CREATE TABLE T1 (F1 INT);
CREATE STORAGE MAP S1 FOR T1 STORE IN TEST_DATA;
CREATE TABLE T2 (F1 INT);
COMMIT;
EXIT;
```

From another session, start a process that updates the database and then waits to be killed:

```
$ SQL$
ATTACH 'FILENAME TEST';
INSERT INTO T2 VALUES (1);
COMMIT;
```

From the first session, update the database with the following sequence:

1. Add rows to a table
2. Drop that table
3. Disconnect
4. Create a new table

```
$ SQL$
ATTACH 'FILE TEST';
INSERT INTO T1 VALUES (1);
INSERT INTO T1 VALUES (2);
INSERT INTO T1 VALUES (3);
INSERT INTO T1 VALUES (4);
COMMIT;

DROP TABLE T1;
```

```
COMMIT;

DISC ALL;

ATTACH 'FILE TEST';
CREATE TABLE T1 (F1 INT);
CREATE STORAGE MAP S1 FOR T1 STORE IN TEST_DATA;
COMMIT;
EXIT;
```

Terminate the second process. Use the DCL STOP/ID command or type ^Y followed by the DCL STOP command so that a DBR process is started. Then attempt to select records from the database:

```
$ SQL$
ATTACH 'FILENAME TEST';
SELECT * FROM T1;
%RDB−E−NO_RECORD, access by dbkey failed because dbkey is no longer associated
with a record
−RDMS−F−NODBK, 57:5:0 does not point to a data record
```

This problem can be avoided by doing any of the following:

- Disabling the PAGE TRANSFER VIA MEMORY and Row Cache features.
- Issuing an RMU/CHECKPOINT command immediately after dropping a table.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.3 RDM$BIND_SNAP_QUIET_POINT Logical Reinstated

Bug 3908414

Over the years, various problems have been reported related to quiet point backups. In particular, database backups and journal backups would sometimes fail with the following error:

```
%RMU−F−TIMEOUT, timeout on quiet
```

Quiet point backups are required so that recovery of a journal can be done without always requiring previous journals. Full database backups can avoid lock conflicts if they wait for the quiet point lock. See the documentation for the RMU/BACKUP commands for more information regarding quiet point backups.

Lock timeout errors for the quiet lock are intended to be returned when a long running update transaction has not completed within the timeout period. However, Oracle Rdb Release 6.0 forced all transactions (read only or read write) to obtain the quiet point lock when starting. That greatly increased the incidence of timeout errors from backups. To remedy this situation, a special logical name was implemented that would force processes starting read only transactions to release the quiet point lock. The logical was "RDM$BIND_SNAP_QUIET_POINT". If that logical was defined to the value "0", then read only transactions would not obtain the quiet point lock.

Defining the RDM$BIND_SNAP_QUIET_POINT logical to "0" would usually resolve problems with timeouts on the quiet lock but it would effectively disable the fast commit performance feature for applications that often switched between read only and read write transactions. (See Bug 884004.) To remedy that situation, the transaction start code was modified to retain the quiet lock during read only transactions but release the lock during the read only transaction if a backup process started. With this change, the

RDM$BIND_SNAP_QUIET_POINT lock logical could be defined without impacting the performance of the fast commit feature. That change was introduced in Releases 7.0.6.3 and 7.1.0.1.

The previous fix resolved performance problems but the logical still could not be defined to "0" if the Hot Standby feature was being used. (See Bug 2656534.) If the Hot Standby feature was being utilized, then the logical had to be undefined or defined to the default value "1". Applications that utilized Hot Standby were still subject to undeserved timeouts from backup commands. In Releases 7.0.7.1 and 7.1.2, the Hot Standby feature was modified to not require read only transactions to hold the quiet point lock. Also, since read only processes would usually release the quiet point lock when a backup started, the RDM$BIND_SNAP_QUIET_POINT logical was completely removed.

After the above changes, for most applications, quiet lock timeouts were no longer an issue. However, a read only transaction would only release the quiet point lock when Oracle Rdb had returned control to the user application. Some complicated queries could execute for an exceptionally long period of time before returning a row to the user application and thus might not release the quiet lock in time to prevent timeout errors for the backup process. For example, read only queries that executed aggregate functions such as SUM or AVERAGE on large sets of rows, or queries that required sorts of large sets of rows before any rows were returned could sometimes not release the quiet point lock before the backup command timed out. (See Bug 3908414.) In addition, any update transaction that started after the backup process requested the quiet point lock would stall until the long running read only request returned control to the user application. That means that it was possible for many database users to stall waiting for the read only transaction to complete even though they had released the quiet point lock.

This release of Oracle Rdb reinstates the RDM$BIND_SNAP_QUIET_POINT logical so that read only transactions can be forced to release the quiet point lock before starting. The logical now has a slightly different meaning than the original implementation. The default value is still "1" but that value now signifies that all transactions will hold the quiet point lock until a backup process requests it. Read only transactions will not obtain the quiet point lock; only read write requests will obtain the quiet point lock. This is the behavior that was introduced in response to Bug 884004. If the logical is defined to be "0", then read only transactions will always release the quiet point lock at the beginning of the transaction, regardless of the existence of a backup process. That implies that all modified buffers in the buffer pool have to be written to disk before the transaction proceeds. Applications that utilize the fast commit feature and often switch between read only and read write transactions within a single attach may experience performance degradation if the logical is defined to "0". This is the behavior that was in place prior to Releases 7.0.6.3 and 7.1.0.1.

Oracle recommends that the RDM$BIND_SNAP_QUIET_POINT logical not be defined for most applications. If the scenario described in Bug 3908414 is being encountered, then the logical can be defined to "0" to force read only transactions to always release the quiet point lock. Rather than define the logical system wide, this logical can be defined for specific jobs that are likely to execute for an extensive period of time before returning to the user application. This parameter may also be manipulated from the RMU/SHOW STATISTICS locking dashboard. That way most users will not have to sacrifice fast commit performance when switching between read only and read write transactions. As of Releases 7.0.7.1 and 7.1.2, Hot Standby is no longer affected by this logical so Hot Standby is not a factor when determining how to define the logical.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.4 Buffer Overflow in Code Generated by RDMS$$STITM_FROM_CACT

Bugs 3354321, 3566244 and 3589230

On rare occasions, an ACCVIO and a bugcheck would occur because of a buffer overflow condition. Code generated by RDMS$$STITM_FROM_CACT may have tried to read beyond its data buffer.

This problem was actually corrected in Oracle Rdb Release 7.1.3.

2.1.5 Sequential Scan Statistics

Bug 3917080

Previously, there was no way to accurately determine the number of strict sequential scans nor the number of DBKEYs returned from those sequential scans.

This problem has been corrected in Oracle Rdb Release 7.1.4. Two new statistics counters record the number of sequential scans started and the number of DBKEYs returned from those sequential scans. These counters are recorded on a database−wide basis (displayed on the "Record Statistics" screen) and on a per−table basis (displayed on the "Logical Area Statistics" screens).

2.1.6 Query with a Shared Predicate in OR Statement Returns Wrong Results

Bug 3918278

The following query with a shared predicate in an OR statment should return two rows.

```
SELECT
   t1.security_id,
   t1.quotation_basis_code
 FROM
   security_quotation t1,
   quotation_basis t2,
   (SELECT c1.security_id, c1.issuer_id FROM equity c1
    union
    SELECT c2.security_id, c2.issuer_id FROM fixed_interest c2) AS t3,
   (SELECT
        c3.security_id,
        c3.chess_eligibility_code AS chess_eligibility_code,
        c3.first_listed_date AS first_listed_date,
        c3.last_listed_date AS last_listed_date
      from
           equity c3 LEFT OUTER JOIN equity_tran c4
           on c3.security_id = c4.security_id
   ) AS t4
 WHERE
    t1.quotation_basis_start_date = '09−SEP−2004'  AND
    t1.quotation_basis_code = t2.quotation_basis_code AND
    t1.security_id          = t3.security_id AND
    t1.security_id          = t4.security_id AND
 (
  ((t4.last_listed_date >= '08−SEP−2004' OR
    t4.last_listed_date = '17−NOV−1858') AND
      t1.quotation_basis_code <> 'RE'
      )
    OR                          ! <== OR statement with shared predicates
  ((t4.last_listed_date >= '09−SEP−2004' OR
    t4.last_listed_date = '17−NOV−1858'  ) AND
```

```
      t1.quotation_basis_code = 'RE'
      )
   OR
  (t4.chess_eligibility_code > 0)
  )
 ;
Tables:
  0 = SECURITY_QUOTATION
  1 = QUOTATION_BASIS
  2 = EQUITY
  3 = FIXED_INTEREST
  4 = EQUITY
  5 = EQUITY_TRAN
Cross block of 3 entries
  Cross block entry 1
    Conjunct: 0.SECURITY_ID = 4.SECURITY_ID
    Conjunct: (((4.LAST_LISTED_DATE >= '8-SEP-2004') OR (4.LAST_LISTED_DATE =
              '17-NOV-1858')) AND (0.QUOTATION_BASIS_CODE <> 'RE')) OR (((
              4.LAST_LISTED_DATE >= '9-SEP-2004') OR (4.LAST_LISTED_DATE =
              '17-NOV-1858')) AND (0.QUOTATION_BASIS_CODE = 'RE')) OR (
              4.CHESS_ELIGIBILITY_CODE > 0)
    Match
      Outer loop
        Merge of 1 entries
          Merge block entry 1
          Match    (Left Outer Join)
            Outer loop
              Get     Retrieval by index of relation 4:EQUITY
                Index name  EQY_U_PRM [0:0]
            Inner loop      (zig-zag)
              Index only retrieval of relation 5:EQUITY_TRAN
                Index name  EQT_U_PRM [0:0]
      Inner loop      (zig-zag)
        Conjunct: 0.QUOTATION_BASIS_START_DATE = '9-SEP-2004'
        Index only retrieval of relation 0:SECURITY_QUOTATION
          Index name  SQB_U_PRM [0:0]
  Cross block entry 2
    Index only retrieval of relation 1:QUOTATION_BASIS
      Index name  QTB_U_PRM [1:1]    Direct lookup
        Keys: 0.QUOTATION_BASIS_CODE = 1.QUOTATION_BASIS_CODE
  Cross block entry 3
    Conjunct: (0.SECURITY_ID = <mapped field>) AND (0.SECURITY_ID =
              4.SECURITY_ID) AND ((4.LAST_LISTED_DATE >= '8-SEP-2004') OR (
              4.LAST_LISTED_DATE >= '9-SEP-2004') OR (4.CHESS_ELIGIBILITY_CODE
               > 0))
    Merge of 1 entries
      Merge block entry 1
      Reduce: <mapped field>, <mapped field>
      Sort: <mapped field>(a), <mapped field>(a)
      Merge of 2 entries
        Merge block entry 1
        Get Retrieval by index of relation 2:EQUITY
          Index name  EQY_U_PRM [1:1] Direct lookup
            Keys: 0.SECURITY_ID = <mapped field>
        Merge block entry 2
        Get     Retrieval by index of relation 3:FIXED_INTEREST
          Index name  FIN_U_PRM [1:1]    Direct lookup
            Keys: 0.SECURITY_ID = <mapped field>
0 rows selected
```

The first appearance of the conjuncts created in the above query represents the complete OR statements.

```
    Conjunct: 0.SECURITY_ID = 4.SECURITY_ID
    Conjunct: (((4.LAST_LISTED_DATE >= '8−SEP−2004') OR (4.LAST_LISTED_DATE =
              '17−NOV−1858')) AND (0.QUOTATION_BASIS_CODE <> 'RE')) OR (((
              4.LAST_LISTED_DATE >= '9−SEP−2004') OR (4.LAST_LISTED_DATE =
              '17−NOV−1858')) AND (0.QUOTATION_BASIS_CODE = 'RE')) OR (
              4.CHESS_ELIGIBILITY_CODE > 0)
```

However, the next appearance is incomplete since the shared predicate "t4.last_listed_date = '17−NOV−1858'" is missing from both the left and right hand side of the OR statement.

```
    Conjunct: (0.SECURITY_ID = <mapped field>) AND (0.SECURITY_ID =
              4.SECURITY_ID) AND ((4.LAST_LISTED_DATE >= '8−SEP−2004') OR (
              4.LAST_LISTED_DATE >= '9−SEP−2004') OR (4.CHESS_ELIGIBILITY_CODE
               > 0))
```

Also missing are the predicates "t1.quotation_basis_code <> 'RE'" and "t1.quotation_basis_code = 'RE'".

The only workaround for this problem is to swap the left and right side terms of the OR statement but this is not acceptable as a real workaround due to the changes to the SQL scripts.

The key parts of this query which contributed to the situation leading to the error are these:

1. The query joins several tables, including derived tables from UNION join and left OUTER join.
2. The WHERE clause contains an OR statement with shared predicates in the left and right side terms.
3. The OR statement also contains a filter predicate on both left and right side terms.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.7 Intermittent Bugchecks on KOD$ROLLBACK or KOD$PREPARE

Bug 3902286

When using SQL*Plus, intermittent bugchecks could occur if lock conflicts were present. This was due to an error in hold cursor processing.

The following shows examples of the bugchecks that could be seen.

```
COSI−F−BUGCHECK, internal consistency failure
Exception occurred  at KOD$ROLLBACK + 00000258
Called from RDMS$$INT_ROLLBACK_TRANSACTION + 0000055C
Called from RDMS$TOP_ROLLBACK_TRANSACTION + 000003EC
Called from BLI$CALLG + 000000BC

COSI−F−BUGCHECK, internal consistency failure
Exception occurred at KOD$PREPARE + 00000228
Called from KOD$COMMIT + 0000018C
Called from RDMS$$INT_COMMIT_TRANSACTION + 000002BC
Called from RDMS$TOP_COMMIT_TRANSACTION + 0000021C
```

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.8 Wrong Result for Query with Constant Column Defined in a View

Bug 1752645

The following query should return 100 rows with NULL for the column "v.const_col".

```
create view mfp_view (employee_id, const_col) as
select e.postal_code, 1 from employees e
  where e.employee_id = any (select z.employee_id from salary_history z );

select employee_id, v.const_col
  from employees left outer join mfp_view v using (employee_id)
where v.const_col is null;
Tables:
  0 = EMPLOYEES
  1 = EMPLOYEES
  2 = SALARY_HISTORY
Conjunct: MISSING (1)
Match    (Left Outer Join)
  Outer loop
    Index only retrieval of relation 0:EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:0]
  Inner loop
    Temporary relation
    Sort: 1.POSTAL_CODE(a)
    Conjunct: <agg0> <> 0
    Match    (Agg Outer Join)
      Outer loop
        Get     Retrieval by index of relation 1:EMPLOYEES
          Index name  EMP_EMPLOYEE_ID [0:0]
      Inner loop      (zig-zag)
        Aggregate-F1: 0:COUNT-ANY (<subselect>)
        Index only retrieval of relation 2:SALARY_HISTORY
          Index name  SH_EMPLOYEE_ID [0:0]
0 rows selected
```

This problem occurs when the main query is an outer join between the EMPLOYEES table and a view using the EMPLOYEE_ID as the join predicate, where the view contains a column of either constant value or an expression of all constant values.

```
create view mfp_view (employee_id, const_col) as
select e.postal_code, 1+2 from employees e
  where e.employee_id = any (select z.employee_id from salary_history z );

select employee_id, v.const_col
  from employees left outer join mfp_view v using (employee_id)
where v.const_col is null;
Tables:
  0 = EMPLOYEES
  1 = EMPLOYEES
  2 = SALARY_HISTORY
Conjunct: MISSING (1 + 2)
Match    (Left Outer Join)
  Outer loop
    Index only retrieval of relation 0:EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:0]
  Inner loop
```

```
    Temporary relation
    Sort: 1.POSTAL_CODE(a)
    Conjunct: <agg0> <> 0
    Match    (Agg Outer Join)
      Outer loop
        Get     Retrieval by index of relation 1:EMPLOYEES
          Index name  EMP_EMPLOYEE_ID [0:0]
      Inner loop      (zig-zag)
        Aggregate-F1: 0:COUNT-ANY (<subselect>)
        Index only retrieval of relation 2:SALARY_HISTORY
          Index name  SH_EMPLOYEE_ID [0:0]
0 rows selected
```

A similar query works if the constant column in the view is concatenated with another column of the EMPLOYEES table, for example middle_initial.

```
create view mfp_view (employee_id, const_col) as
select e.postal_code, '1'||e.middle_initial from employees e
  where e.employee_id = any (select z.employee_id from salary_history z );

select employee_id, v.const_col
  from employees left outer join mfp_view v using (employee_id)
where v.const_col is null;
Tables:
  0 = EMPLOYEES
  1 = EMPLOYEES
  2 = SALARY_HISTORY
Conjunct: MISSING ('1' || 1.MIDDLE_INITIAL)
Match    (Left Outer Join)
  Outer loop
    Index only retrieval of relation 0:EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:0]
  Inner loop
    Temporary relation
    Sort: 1.POSTAL_CODE(a)
    Conjunct: <agg0> <> 0
    Match    (Agg Outer Join)
      Outer loop
        Get     Retrieval by index of relation 1:EMPLOYEES
          Index name  EMP_EMPLOYEE_ID [0:0]
      Inner loop      (zig-zag)
        Aggregate-F1: 0:COUNT-ANY (<subselect>)
        Index only retrieval of relation 2:SALARY_HISTORY
          Index name  SH_EMPLOYEE_ID [0:0]
 EMPLOYEE_ID   I.CONST_COL
 00164         NULL
 00165         NULL
  ...etc...
 00471         NULL
100 rows selected
```

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.9 NOREQIDT Error After Many Attaches/Detaches

Bug 3971379

An application that utilized lock timeouts (for example, the WAIT clause of the SET TRANSACTION statement) and often disconnected from a database could eventually encounter the following error:

```
%RDMS−F−NOREQIDT, reached internal maximum number of simultaneous timer requests
```

This problem could be reproduced by creating two login sessions. The first session would lock a row in a table. The second session would repeatedly attempt to access the same row. After each attempt, it would disconnect from the database and attach again. For example:

SESSION 1:

```
SQL> ATTACH 'FILE MF_PERSONNEL';
SQL> UPDATE EMPLOYEES SET SEX = SEX WHERE EMPLOYEE_ID = '00164';
```

SESSION 2:

Repeat the following sequence of commands without exiting SQL:

```
SQL> ATTACH 'FILENAME MF_PERSONNEL';
SQL> SET TRANSACTION READ WRITE WAIT 1;
SQL> UPDATE EMPLOYEES SET SEX = SEX WHERE EMPLOYEE_ID = '00164';
SQL> ROLLBACK;
SQL> DISCONNECT ALL;
```

After about 100 iterations, the UPDATE command would fail with the NOREQIDT error.

This problem can be avoided by not repeatedly detaching from and attaching to the database within a single invocation of a database application, or by not using the lock timeout feature.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.10 Processes Hang in HIB State

Bugs 2881846 and 3807295

Oracle Rdb applications that hibernate may occasionally hang in HIB state if after−image journaling (AIJ) is enabled. Note that applications utilizing the OpenVMS POSIX Threads Library will often hibernate, so threaded applications are especially vulnerable to this issue.

This problem could occur if a user application had executed the OpenVMS $HIBER system service and one of the following Oracle Rdb events occurred while the process was hibernating:

- A Global Checkpoint request was issued by a journal switch or an RMU/CHECKPOINT command. (This issue was resolved in releases 7.0.7.2 and 7.1.2.2; see Bug 2881846 for more information.)
- A checkpoint timer expired.
- A two−phase commit (2PC or DECdtm) transaction that involved the process was ended.

Note that if the OpenVMS POSIX Threads Library is being utilized, the OpenVMS remedial kit VMS732_SYS−V0600 may also be required to completely resolve this issue.

This problem can be avoided by disabling journaling. Oracle does not recommend disabling journaling unless another suitable database recovery method is available.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.11 Missing Records in Dynamic FAST FIRST Shortcut Termination

Bug 3983608

When the optimizer chose to use a dynamic retrieval strategy, and the type of dynamic tactic chosen was FAST FIRST, it was possible that Rdb would fail to deliver rows that should have been selected by the query.

This problem was introduced in Oracle Rdb Release 7.1.3 with the introduction of the FAST FIRST shortcut termination feature.

If a query using the FAST FIRST tactic delivers less than 1026 rows during the first index scan, and if more than 1024 dbkeys are read from that index and the index scan completes, FAST FIRST shortcut termination will be employed. In this case, it is possible that the 1025th dbkey read from the index will not be delivered. If FAST FIRST is running when the first index scan completes, the 1025th dbkey and every 1024 dbkeys after that, can fail to be delivered.

The following example shows a query that should return 1025 rows but only returns 1024.

```
SQL> select count(*) from t1 where f1=1;
        1025
1 row selected
SQL> set flags 'strategy,exec'
SQL> select * from t1 where f1=1;
~E#0000.00(2) Estim   Index/Estimate 1/2
~E#0000.00(2) BgrNdx1 EofData  DBKeys=2  Fetches=0+0  RecsOut=0 #Bufs=1
~E#0000.00(2) Fin     Buf      DBKeys=2  Fetches=0+1  RecsOut=2
~S#0001
Leaf#01 FFirst T1 Card=1025
  BgrNdx1 I1 [1:1] Fan=17
~E#0001.01(1) Estim   Index/Estimate 1/2050
          F1            F2
           1             1
.
.
.
           1             1
~E#0001.01(1) BgrNdx1 EofBuf   DBKeys=1024  Fetches=0+3  RecsOut=1024
~E#0001.01(1) BgrNdx1 EofData  DBKeys=1025* Fetches=0+0  RecsOut=1024 #Bufs=10
~E#0001.01(1) FgrNdx  FFirst   DBKeys=1024  Fetches=0+12  RecsOut=1024`ABA
~E#0001.01(1) Fin     TTblIni  DBKeys=0  Fetches=0+0  RecsOut=1024`ABA
           1             1
1024 rows selected
```

In this example, the last execution trace line shows that the final (Fin) phase of execution is being abandoned. This indicates FAST FIRST shortcut termination is being employed.

During FAST FIRST query execution, the first background index is scanned and the dbkeys from the index are passed to foreground. Foreground processing fetches the row, tests all conditions in the query against the row, and if necessary delivers the row. During the phase, the 1025th row and every subsequent 1024th row were not being passed to foreground.

If more than 1024 dbkeys are delivered, foreground is abandoned and background's dbkey list is used to deliver the remaining rows. In this case, the rows not passed to foreground will be fetched from background's dbkey list and delivered during the final (Fin) phase, and therefore all rows will be delivered.

The problem can be avoided in several ways including:

- Optimizing the query for TOTAL TIME rather than FAST FIRST.
- Enabling the Bitmapped Scan feature. To enable bitmap scan, at least one of the indexes used by the query must be of *TYPE IS SORTED RANKED*.
- Disabling the dynamic optimizer using the *MAX_STABILITY* flag or logical name.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.12 DBR Bugchecks at DBR$DDTM_RESOLVE + 000005A8

Bug 4006404

If a process was involved in a two−phase commit (2PC of DECdtm) transaction and it failed after the prepare phase of a transaction but before the commit phase, and the database did not have the FAST COMMIT feature enabled, it was possible for the database recovery process (DBR) to fail with the following exception:

```
***** Exception at 00071858 : DBR$DDTM_RESOLVE + 000005A8
%COSI−F−BUGCHECK, internal consistency failure
```

The problem was introduced in Oracle Rdb Releases 7.0.7.1 and 7.1.2.

Enabling the FAST COMMIT feature will prevent this problem.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.13 DBR Bugchecks at DBR$WAKE_ALL + 000000F4

Bug 4006404

It was possible for a database recovery process (DBR) to fail with the following exceptions:

```
***** Exception at 00088C54 : DBR$WAKE_ALL + 000000F4
%COSI−F−UNEXPERR, unexpected system error
−SYSTEM−F−REMRSRC, insufficient system resources at remote node

***** Exception at 00088C54 : DBR$WAKE_ALL + 000000F4
%COSI−F−UNEXPERR, unexpected system error
−SYSTEM−F−NOSUCHTHREAD, specified kernel thread does not exist
```

The DBR did not anticipate the possibility of those errors occurring when issuing the OpenVMS $WAKE system service and would terminate when those error status codes were returned by the system service.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.14 Database Corruption when 2PC Transaction Fails

Bug 4011078

It was possible for a database to become corrupt when the following conditions were true:

- The FAST COMMIT feature was enabled.
- A failing process was a participant in a two−phase commit (2PC) transaction.
- The process failed after completing the "prepare" phase of a 2PC transaction but before completing the "commit" phase.
- Other participants in the same transaction failed before completing the "prepare" phase.

In this situation, the database recovery process (DBR) would neglect to rollback the failed transaction.

This problem was introduced in Oracle Rdb Releases 7.0.7.1 and 7.1.2.

There is no workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.15 Hangs With No Stall Messages

Bug 4027315

When Oracle Rdb would call RMS for asynchronous operation, if the call to RMS returned an error, Oracle Rdb would hang waiting for a completion AST from RMS. Often, no stall message would be displayed by RMU/SHOW STATISTICS.

For example, if Oracle Rdb was using a temporary file on disk, and it did not have sufficient ASTLM quota, it could hang when attempting to read from or write to the temporary file. Oracle Rdb may have issued many asynchronous disk writes (ABW) to flush modified page buffers to disk, consuming all available ASTLM quota. While that I/O was underway, if Oracle Rdb attempted to access a temporary RMS file, it would not properly handle the error returned by RMS and would hang.

Oracle Rdb will now properly trap RMS errors and report them to the application. For example, the following error may be returned if quota is exhausted:

```
%RDB−F−IO_ERROR, input or output error
−COSI−F−READERR, read error
−RMS−F−CDA, cannot deliver AST
```

This problem can be avoided by ensuring that users have sufficient DIOLM and ASTLM quota to concurrently flush all allocated page buffers, while also having enough additional quota to do I/O to other files that are used by Oracle Rdb such as journals, temporary files, etc. For example, if a process is using 500 page buffers, the minimum quota values for DIOLM and ASTLM would be greater than 500.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.16 Transaction State Not Set in Root

If the COMMIT TO JOURNAL OPTIMIZATION feature was enabled, the transaction state for database users was not correctly shown by the RMU/DUMP/USERS command. That is, RMU/DUMP/USERS may have indicated that a process did not have a transaction active when in fact it did. This problem would occur because the database root file data structures that reflected the transaction state of a user were not always updated when the COMMIT TO JOURNAL OPTIMIZATION feature was enabled.

This problem has been corrected in Oracle Rdb Release 7.1.4. The database root file will now correctly show that a user has a transaction active. Note that when the COMMIT TO JOURNAL OPTIMIZATION feature is enabled, the transaction sequence number (TSN) displayed by RMU/DUMP/USER will be zero. The TSN is not maintained in the database root file when this optimization is being used.

2.1.17 Rdb Returns %RDB−E−EXCESS_TRANS

Bug 4104994

Applications occasionally received an %RDB−E−EXCESS_TRANS error seemingly at random in response to requests such as opening or fetching from a cursor. This problem was experienced after an upgrade to Rdb 7.1.3 in an Oracle Forms application which uses SQL/Services 7.1.5.9.1 but it is not peculiar to that combination.

The problem was caused by the use of uninitialized stack memory and cannot be readily reproduced.

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.18 Cursor Fetch Returns Wrong Result in Second Execution

Bugs 4086431, 2882908, and 1329838

Customer uses cursors to fetch the next record of a given list. This normally works fine but sometimes the first fetch after opening the cursor returns the wrong record.

The following is the simple reproducer of the problem.

```
! the first table contains only 2 rows
!
SELECT SLU_ID,SET_ID,UNDR_SECU FROM TP_INST;
 SLU_ID        SET_ID   UNDR_SECU      INST_TYP
 CBON               2   DE0001135002   BON
 NBON            1007   DE4444444444   BON
2 rows selected

! the second table also contains 2 rows
!
SELECT ISIN,SET_ID FROM tt_trdd_inst;
 ISIN                SET_ID
 DE0001135002             2
 DE4444444444          1007
2 rows selected
```

```
! Declare the host variables and cursor
!
SET TRANSACTION;
DECLARE :SLU_ID CHAR (5);
DECLARE :SET_ID INTEGER;
DECLARE :UNDR_SECU CHAR (12);

DECLARE Cursor_bug CURSOR FOR
SELECT * FROM (
  SELECT T1.SLU_ID, T1.SET_ID, T1.UNDR_SECU  FROM
     TT_TRDD_INST, TP_INST
  WHERE T1.UNDR_SECU = T2.ISIN AND
        T1.SET_ID    = T2.SET_ID AND
        INST_TYP     = 'BON'
  ORDER BY T1.SLU_ID, T1.SET_ID, T1.UNDR_SECU
        ) as TT
WHERE
       (
        ( TT.SLU_ID > :SLU_ID )
                OR
        ( ( TT.SLU_ID = :SLU_ID ) AND
          ( TT.SET_ID > :SET_ID ) )
                OR
        ( ( TT.SLU_ID = :SLU_ID ) AND
          ( TT.SET_ID = :SET_ID ) AND
          ( TT.UNDR_SECU > :UNDR_SECU ) )
                )
commit;

! Define the values of the host variables
!
BEGIN
    SET :SLU_ID = 'CBON';
    SET :SET_ID = 2;
    SET :UNDR_SECU = 'DE0001134997';
END;

! Open the cursor the first time
OPEN Cursor_bug;

! It should fetch the first row 'CBON'
FETCH Cursor_bug;
 SLU_ID        SET_ID   UNDR_SECU
 CBON               2   DE0001135002

commit;

! Repeating the above OPEN/FETCH a second time gives the wrong result
! 'NBON' instead 'CBON'

SET TRANSACTION;
BEGIN
    SET :SLU_ID = 'CBON';
    SET :SET_ID = 2;
    SET :UNDR_SECU = 'DE0001134997';
END;

OPEN Cursor_bug;
```

```
! It should fetch the first row 'CBON'
FETCH Cursor_bug;
 SLU_ID        SET_ID   UNDR_SECU
 NBON            1007   DE4444444444

commit;
```

The key parts of this cursor query which contributed to the situation leading to the error are these:

1. The cursor query contains the outer SELECT statement as a derived table wrapped around the inner SELECT statment with two tables.
2. The inner SELECT statement contains a WHERE clause with equality predicates joining the two tables and a filter predicate followed by an ORDER BY clause.
3. The outer SELECT statement contains a WHERE clause with two complex OR predicates which reference the columns via the derived table.

As a workaround, when the WHERE clauses within the cursor are defined in a different order, the cursor works fine in all cases.

```
! Swap the first and third expressions under the OR predicates

DECLARE Cursor_bug CURSOR FOR
SELECT * FROM (
  SELECT T1.SLU_ID, T1.SET_ID, T1.UNDR_SECU  FROM
     TT_TRDD_INST, TP_INST
  WHERE T1.UNDR_SECU = T2.ISIN AND
        T1.SET_ID    = T2.SET_ID AND
        INST_TYP     = 'BON'
  ORDER BY T1.SLU_ID, T1.SET_ID, T1.UNDR_SECU
        ) as TT
WHERE
       (
        ( ( TT.SLU_ID = :SLU_ID ) AND
          ( TT.SET_ID = :SET_ID ) AND
          ( TT.UNDR_SECU > :UNDR_SECU ) )
                OR
        ( ( TT.SLU_ID = :SLU_ID ) AND
          ( TT.SET_ID > :SET_ID ) )
                OR
        ( TT.SLU_ID > :SLU_ID )
                )
commit;
```

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.1.19 RMONSTART71 Corrected Use of Parameter "/RESIDENT"

Bug 4115074

In prior releases of Oracle Rdb, the P1 parameter to RMONSTART71.COM was documented as accepting a string "/RESIDENT" to cause various Oracle Rdb images to be installed as resident images. However, the processing of the P1 parameter was not correct and would generally not work correctly when "/RESIDENT" was included.

This problem has been corrected in Oracle Rdb Release 7.1.4. Including "/RESIDENT" in the P1 parameter now correctly causes images to be installed resident.

On OpenVMS Alpha systems, application performance may improve by installing Oracle Rdb images as "resident" with the OpenVMS Install utility (INSTALL). Installing images as resident allows them to take advantage of the OpenVMS Alpha image−slicing features.

The code sections of an image installed as resident reside in huge pages called granularity hint regions (GHRs) in memory. The OpenVMS Alpha hardware can consider a set of pages as a single GHR. This GHR can be mapped by a single page table entry (PTE) in the translation buffer (TB). The result is a reduction in TB miss rates. For more information on slicing shareable images, see the OpenVMS documentation set.

When RAD (OpenVMS Resource Affinity Domains) support is enabled, VMS can replicate /RESIDENT installed image data on each RAD. The advantage to this replication is that CPU access to the image memory will always be in the same RAD. This "on−RAD" memory access is desirable for highest performance.

2.2 SQL Errors Fixed

2.2.1 Illegal Use of PARTITION Clause Not Detected by SET TRANSACTION

Bug 3829329

In prior versions of Oracle Rdb, the SET TRANSACTION ... RESERVING clause allowed a view to be specified with the PARTITION clause. This is not permitted and Rdb should have reported an error. In fact, the PARTITION clause was ignored.

Oracle Rdb now diagnoses this error and reports an error as shown in this example.

```
SQL> set transaction read write
cont>     reserving EMP_VIEW partition (2) for shared read;
%RDB−E−BAD_TPB_CONTENT, invalid transaction parameters in the transaction
parameter block (TPB)
−RDMS−F−TABLENOMAP, table "EMP_VIEW" is not partitioned with a storage map
```

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.2.2 Unexpected Bugcheck when RENAME TO Clause with Column

Bug 3938814

In prior versions of Oracle Rdb, attempts to use the RENAME TO clause for a column definition produced a bugcheck instead of reporting an error. The following example shows this error:

```
SQL> alter table T_T_T
cont>    alter column aaa
cont>        rename to A_A_A
cont> ;
%RDMS−I−BUGCHKDMP, generating bugcheck dump file USER2:[TESTING]SQLBUGCHK.DMP;
%SYSTEM−F−ACCVIO, access violation, reason mask=00,
virtual address=0000000000000000, PC=00000000001F6E28, PS=0000001B
```

This problem has been corrected in Oracle Rdb Release 7.1.4. Although the RENAME TO clause is planned for a future release, it is not currently supported and should have been rejected by SQL. The following example shows the new diagnostic.

```
SQL> alter table T_T_T
cont>    alter column aaa
cont>        rename to A_A_A
cont> ;
%SQL−F−WISH_LIST, Feature not yet implemented − RENAME column
```

2.2.3 Exception at 0009AEB0: GEM_DA_WALK_DMN_FOREST + 000018D0

Bug 3947784

SQL$PRE and SQL$MOD would fail and produce a SQLBUGCHK.DMP file if run with the /MACHINE_CODE qualifier and without the /LIST qualifier. This occurred with later versions of Rdb 7.1. The /LIST qualifier is required if the /MACHINE_CODE qualifier is specified. However in earlier versions of Oracle Rdb, the /MACHINE_CODE qualifier was simply ignored if /LIST was omitted. With later versions of Oracle Rdb 7.1, both SQL$MOD and SQL$PRE fail if /LIST is omitted.

For example, the following SQL$PRE command line would result in the failure shown:

```
VMS> SQL$PRE/COBOL/MACHINE SQL$SAMPLE:SQL_REPORT.SCO
Assertion failure:  Compiler internal error − please submit problem report
%RDMS−I−BUGCHKDMP, generating bugcheck dump file USERS:[MY_HOME]SQLBUGCHK.DMP;
```

The SQLBUGCHK.DMP file would contain the following signature:

```
***** Exception at 0009C3B0 : GEM_DA_WALK_DMN_FOREST + 000018D0
%GEM−F−ASSERTION, Compiler internal error − please submit problem report
```

The problem has now been fixed so that SQL$PRE and SQL$MOD will produce the following message instead of failing:

```
VMS> SQL$PRE/COBOL/MACHINE SQL$SAMPLE:SQL_REPORT.SCO
%SQL−F−NOLISTQUAL, /MACHINE qualifier requires /LIST
```

As a workaround, include the /LIST qualifier when specifying the /MACHINE_CODE qualifier as required.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.2.4 Repeated CREATE of a LOCAL TEMPORARY Table Would Bugcheck

Bug 3953460

In prior versions of Oracle Rdb, attempts to rollback a LOCAL TEMPORARY table definition and re−create it would bugcheck. The following example shows this problem.

```
SQL> create local temporary table tt_tmp (f1 integer, f2 char(67))
cont> on commit preserve rows;
SQL> insert into tt_tmp values (12200,'Accounting');
1 row inserted
SQL> rollback;
SQL> create local temporary table tt_tmp (f1 integer, f2 char(67))
cont> on commit preserve rows;
SQL> insert into tt_tmp values (12200,'Accounting');
%RDMS−I−BUGCHKDMP, generating bugcheck dump file USER2:[TESTING]RDSBUGCHK.DMP;
```

This problem has been corrected in Oracle Rdb Release 7.1.4. Rdb now correctly removes the old name from the module specific symbol table.

2.2.5 Exception at 00710BE8: SQL$$SUBMIT_COM_FILE + 00001658

Bug 3876550

Under some circumstances, if an Oracle RDBMS user environment is defined for a user who runs SQL$MOD, it would fail and produce a SQLBUGCHK.DMP file if the /OBJECT, /LIST, or /DIAGNOSTIC qualifier were used. This problem affects SQL$PRE as well.

This occurred because the ORAUSER.COM command file prefixes the list of logical name tables defined in LNM$FILE_DEV with the name of a logical name table in which the user did not have permission to create a new logical name.

For example, after running the ORAUSER.COM, the user's LNM$FILE_DEV might look similar to the following:

```
VMS> SHOW LOGICAL/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV
   "LNM$FILE_DEV" = "SERVER_1414322382" (LNM$PROCESS_DIRECTORY)
        = "LNM$PROCESS"
        = "LNM$JOB"
        = "LNM$GROUP"
        = "LNM$SYSTEM"
```

In the above example, "SERVER_1414322382" is a logical name table used by the Oracle RDBMS user environment to define various user environment logical names. If the user did not have permission to create a new logical name in the SERVER_1414322382 table, using the /OBJECT, /LIST, or /DIAGNOSTIC qualifier on a SQL$MOD or SQL$PRE command would cause a SQLBUGCHK.DMP file to be created with the following signature:

```
***** Exception at 00710BE8 : SQL$$SUBMIT_COM_FILE + 00001658
%COSI−F−UNEXPERR, unexpected system error
−SYSTEM−F−NOPRIV, insufficient privilege or object protection violation
```

The problem has now been fixed so that SQL$PRE and SQL$MOD will complete normally.

As a workaround, prior to doing SQL$MOD or SQL$PRE compiles, the Oracle command file REMORACLE.COM can be run. This removes the Oracle user environment and deletes the server−instance related logical name table from LNM$FILE_DEV. If it is necessary to have the Oracle RDBMS user environment while running SQL$MOD or SQL$PRE, then a command file with the following commands can be run in the user environment:

```
$ ora_install_id = f$trnlm("ora_install_id")
$ if ora_install_id .eqs. "" then exit
$ lnm = f$trnlnm("LNM$FILE_DEV","LNM$PROCESS_DIRECTORY",0)
$ if lnm .nes. ora_install_id then exit
$ max_idx = −
        f$trnlnm("LNM$FILE_DEV","LNM$PROCESS_DIRECTORY",,,,"MAX_INDEX")
$ if max_idx .ge. 1
$ then
$     lnmstr = f$trnlnm("LNM$FILE_DEV","LNM$PROCESS_DIRECTORY",1)
$     lnmstr = lnmstr+","
$ else
$     lnmstr = "LNM$PROCESS,"
$ endif
$ lnmstr = lnmstr+ora_install_id
$ if max_idx .gt. 1
$ then
$     idx = 2
$     lnm_loop:
$         lnm = f$trnlnm("LNM$FILE_DEV","LNM$PROCESS_DIRECTORY",idx)
$         idx = idx + 1
$         lnmstr = lnmstr+","
$         lnmstr = lnmstr+lnm
$         if idx .le. max_idx then goto lnm_loop
$ endif
$ define/nolog/table=lnm$process_directory lnm$file_dev 'lnmstr'
$ exit
```

The above DCL procedure will edit LNM$FILE_DEV so that the Oracle logical name table is after the LNM$PROCESS table. This will allow SQL$MOD and SQL$PRE to succeed when using the /OBJECT, /LIST, or /DIAGNOSTIC qualifiers.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.2.6 ALTER DOMAIN Can Lead to Corrupted Domain Definition

Bug 3899502

In prior versions of Oracle Rdb V7.1, it was possible to create a view which referenced a domain and if the domain had either a DEFAULT clause or a CHECK constraint then it was possible for a subsequent metadata operation to corrupt that domain definition.

When a subsequent ALTER DOMAIN was executed, the view's reference to the CHECK constraint or DEFAULT value was incorrectly using the references from the domain's definitions. That is, the LIST OF BYTE VARYING column values (RDB$DEFAULT_VALUE2 and RDB$VALID_BLR) are shared between this view and the domain. This is incorrect behavior: a LIST OF BYTE VARYING pointer should never be shared by different rows in a database. A subsequent DROP VIEW will erase the LIST OF BYTE VARYING leaving the domain corrupted.

Once corrupted, references to the domain by DROP DOMAIN or ALTER TABLE will result in an error or, in some cases, a bugcheck.

The following example shows the problem:

```
$ on error then continue
$ sql$
create database
    filename abc;

set dialect 'sql99';

create domain DD
    integer
    default 99
    check (value > 0)
;

show domain DD;
DD                              INTEGER
```

```
 Oracle Rdb default: 99
 Valid If:       (value > 0)

create table TT
    (a DD);

create view VV (a)
    as select a from TT;

alter domain DD
    double precision;

show domain DD;
DD                              DOUBLE PRECISION
 Oracle Rdb default: 99
 Valid If:       (value > 0)

commit;
disconnect all;
$
$ ! show the database is clean
$ rmu/verify/segmented/larea=*/nolog abc
$
$ sql$
attach 'filename abc';
-- note it is the COMMIT that actually erases the data
drop view VV;
commit;
show domain DD;
DD                              DOUBLE PRECISION
%RDB-E-NO_RECORD, access by dbkey failed because dbkey is no longer associated
with a record
-RDMS-F-NODBK, 1:613:12 does not point to a data record
create table SS
    (a DD);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-INVALID_BLR, request BLR is incorrect at offset 3
rollback;
disconnect all;
$
$ ! show the database has a problem
$ rmu/verify/segmented/larea=*/nolog abc
%RMU-E-ERRSEGFET, Error fetching segmented string's primary segment.
%RMU-I-SEGSTRDBK, Segmented string is at logical dbkey 1:613:10.
%RMU-I-SEGRECDBK, Data record is at logical dbkey 6:439:1.
%RMU-E-ERRSEGFET, Error fetching segmented string's primary segment.
%RMU-I-SEGSTRDBK, Segmented string is at logical dbkey 1:613:12.
%RMU-I-SEGRECDBK, Data record is at logical dbkey 6:439:1.
$
$ sql$
drop database
    filename abc;
$
$ set noverify
```

This problem has been corrected in Oracle Rdb Release 7.1.4. However, if errors such as those shown in the example are observed, please contact Oracle Support.

2.2.7 SQL Precompiler Loops Compiling a C Program

Bug 4068824

In some cases, the SQL Precompiler would go into a loop processing a C program. In most cases this is caused by code between the function signature and the "{" for the function block but in some cases the condition is triggered by code which does not follow that pattern.

For example, each of the following functions would trigger a loop of the SQL precompiler:

```
static short fn (arg1)
#define ZERO_L l=0
{
    ZERO_L;
    return 1;
}
static short fn1 (arg1)
// C++ style comment (allowed in ANSI C)
{
    ZERO_L;
    return 1;
}
static short fn3 (arg1)
\* bad comment */
{
    ZERO_L;
    return 1;
}
```

In each of the above cases, SQL$PRE encounters something it doesn't expect between the function signature and the "{" that opens the function body. In the first case, it is "#"; in the second, "//"; and in the third case, a "\". Note that only in the last example would the OpenVMS C compiler generate a syntax error.

In addition, some programs with nothing between the code function signature and "{" still trigger the loop. For example, the following program would cause SQL$PRE to loop:

```
include <string.h>
typedef struct ab_entry_struct
{
    int a;
    int b;
} AB_ENTRY;
AB_ENTRY  _align(quadword) *read_queue_entry;
main( int argc, char *argv[] )
{
char shutd_true[10], shutd_value[10];
          {
              ;
          }
    if (memcmp(shutd_true, shutd_value, sizeof(shutd_value)) == 0)
    {
    }
}
```

In the above example, SQL$PRE does not recognize the "_align" keyword and misinterprets the rest of the program. In versions prior to Rdb 7.1.2, this resulted in SQL$PRE missing many variable declarations but did not trigger a loop.

The problem has now been fixed so that SQL$PRE will not loop. In addition, SQL$PRE has been enhanced to recognize the following additional OpenVMS C keywords: "_align", "__align", "_Bool", "_Complex", "__forceinline', "_Imaginary", "__inline", "inline", "__restrict", "signed", "__unaligned", and "wchar_t".

As a workaround, for the cases in the first code example, remove the code between the function signature and the "{". For the second code example, there is no workaround.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.2.8 SQL Precompiler Bugchecks On Syntax Error − SQL$$WALK_VALUE_LIST + 000002DC

Bug 3947784

If a precompiled SQL or SQL module language program contained a SELECT statement with an EDIT USING clause, SQL$PRE and SQL$MOD would fail and produce a SQLBUGCHK.DMP file. The EDIT USING clause is only allowed in interactive SQL but the appropriate response is an error message.

For example, if the following SELECT statement appeared in a precompiled SQL program (say, as part of a cursor definition), SQL$PRE would bugcheck:

```
   SELECT birthday as bd edit using 'dd/nnbrr:pp'
      FROM employees
      WHERE employee_id = '00120';
```

The SQLBUGCHK.DMP file would contain the following signature:

```
***** Exception at 004BC95C : SQL$$WALK_VALUE_LIST + 000004DC
%SQL−F−BUGCHK, There has been a fatal error. Please contact your Oracle support
representative.
```

The problem has now been fixed so that SQL$PRE and SQL$MOD will produce the following message instead of failing:

```
  SELECT birthday as bd edit using 'dd/nnbrr:pp'
                        1
%SQL−E−EDITUSING, (1) The EDIT USING clause may only be used in interactive SQL
```

As a workaround, do not use the EDIT USING clause except in interactive SQL which is the only context in which it is supported.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3 RMU Errors Fixed

2.3.1 RMU/ALTER DISPLAY ROOT USER Command Returns Errors if Multiple Database Users

Bug 3829090

There was a problem in the RMU/ALTER DISPLAY ROOT USER command where a bad parameter was passed to a message which put out a blank line between multiple active database user entries. The VMS error messages were not serious but just related to the bad message parameter. The user display continued despite the VMS errors returned.

The following example shows the errors put out by the RMU/ALTER DISPLAY ROOT USERS command between multiple active database user entries.

```
RdbALTER> display root user
Active user 0
    Process ID is 33A00158
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 13
    Recovery journal filename is
%RMU−I−BLANK_1,
−SYSTEM−S−BADATTRIB, bad attribute control list
−SYSTEM−W−ILLPAGCNT, illegal page count parameter
−FORMS−S−NOMSG, Message number 00D5A789
−SECPACK−W−NOMSG, Message number 00D994F8
−CLI−W−NOMSG, Message number 00031E68
−SYSTEM−S−NORMAL, normal successful completion
−SYSTEM−S−NORMAL, normal successful completion
−SYSTEM−S−NORMAL, normal successful completion
Active user 1
    Process ID is 33A00157
    Stream ID is 1
    Monitor ID is 1
    Transaction ID is 655
    Recovery journal filename is
DISK:[DIRECTORY]PERSONNEL$00016401C103.RUJ;1
RdbALTER>
```

As a workaround for this problem, the RMU/DUMP/USERS command can be used which also displays active database users.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.2 RMU/VERIFY Access Violation When Readying an Invalid Logical Area

Bug 3776327

An access violation occurred in RMU/VERIFY if a bad logical area id number was detected because of database corruption while RMU/VERIFY was readying logical areas when verifying database data pages. RMU/VERIFY correctly reported the bad logical area id but when RMU/VERIFY attempted to continue the database verification, it failed to check if a bad storage area id number had been gotten based on the bad logical area id. The bad storage area id was then used to get the address of the storage area entry in the database root. This caused the access violation. This has been corrected.

The following example shows the problem.

```
$RMU/VERIFY/ALL/LOG BAD_DB

%RMU−I−BGNSEGPAG, beginning verification of RDB$SYSTEM storage area
%RMU−W−CANTFINDLAREA, cannot locate logical area 312 in area inventory page list
%RMU−E−BDLAREADY, error readying logical area with dbid 312
%SYSTEM−W−ACCVIO, access violation, reason mask=00, virtual address=000000000000
000B, PC=00000000003BB3E8, PS=0000001B

%RMU−F−ABORTVER,  fatal error encountered; aborting verification
%SYSTEM−F−ACCVIO, access violation, reason mask=00, virtual address=000000000000
000B, PC=00000000003BB3E8, PS=0000001B
%RMU−F−FATALOSI, Fatal error from the Operating System Interface.
%RMU−I−BUGCHKDMP, generating bugcheck dump file
SERDB_USER1:[HOCHULI]RMUBUGCHK.DMP;
```

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.3 RMU/BACKUP Access Violation When Backing Up the Root ACL

Bug 3866383

An access violation could occur during the RMU/BACKUP of an Oracle Rdb database when the database root ACL (Access Control List) was being backed up. This happened because of a problem testing for the end of the memory allocated to hold the ACL. This problem only occurred when certain memory boundary conditions were reached.

The following example shows the access violation occurring during an RMU/BACKUP when the database ACL was being backed up.

```
$rmu/backup/online/log rdms_db bckdir:rdms_db.rbf
 %RMU−I−QUIETPT, waiting for database quiet point at 16−AUG−2004 02:45:10.65
 %RMU−I−RELQUIETPT, Database quiet point lock has been released at
        16−AUG−2004 02:45:10.81.
 %SYSTEM−F−ACCVIO, access violation, reason mask=00, virtual
        address=0000000000CF2000, PC=00000000003E9620, PS=0000001B
  %RMU−F−FATALERR,  fatal error on BACKUP
  %RMU−F−FTL_BCK, Fatal error for BACKUP operation at 16−AUG−2004 02:45:10.91
```

This problem can be avoided by using the /NOACL qualifier on the backup and recreating the database root ACL after the restore.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.4 RMU/CONVERT of Database Already at Current Version Left AIJ Disabled

Bug 3729079

If an RMU/CONVERT was performed on a database that was already at the current Rdb version (so that no conversion was necessary), the after image journaling, which was temporarily disabled during the RMU/CONVERT, was never re−enabled by RMU. Now, for the RMU/CONVERT of a database where there is no conversion work to do since the database is already at the current level, the after image journaling will not be disabled by RMU so there will be no need to re−enable it. Note that this does not include the /COMMIT or /ROLLBACK of a conversion already made from a previous version of Rdb where /NOCOMMIT was specified. In this case, the database is either being permanently converted to the current version or it is being rolled back to a previous version.

The following example shows that if an RMU/CONVERT was performed on a database that was already at the current Rdb version and the RMU−W−NOCVTCOM message was output by RMU, after image journaling, if it had been enabled for the database, was not reenabled by RMU.

```
$ rmu/convert mf_personnel
%RMU−I−RMUTXT_000, Executing RMU for Oracle Rdb V7.1−241
Are you satisfied with your backup of
DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1
and your backup of any associated .aij files [N]? y
%RMU−I−AIJ_DISABLED, after−image journaling is being disabled temporarily for
the Convert operation
%RMU−I−LOGCONVRT, database root converted to current structure level
%RMU−W−NOCVTCOM, Database DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1
is already at the current structure level.

$RMU/DUMP/HEADER DEVICE:[DIRECTORY]MF_PERSONNEL

    AIJ Journaling...
      − After−image journaling is disabled
```

If this problem occurs, after image journaling must be re−enabled by the user.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.5 RMU/LOAD/PARALLEL Command Could Hang if it was Repeated

Bug 3803694

The RMU/LOAD/PARALLEL command could hang if it was repeated one or more times. This happened because an event flag used to coordinate communication between the root process and the executor processes was not properly initialized. This problem is now fixed.

The following example shows the parallel load hang when the load was repeated.

```
$ rmu/unload  [.db]testdb  tab1  tab1.unl
%RMU−I−DATRECUNL,  6 data records unloaded.
```

```
$ rmu/load/log/parallel=(buf=50,exec=2) −    ! normal successful completion
    [.db]testdb  tab1  tab1.unl
%RMU−I−EXECUTORMAP, Executor EXECUTOR_1 (pid: 2F03372D) will load storage
area AREA1.
%RMU−I−EXECUTORMAP, Executor EXECUTOR_2 (pid: 2F03292E) will load storage
area AREA2.
EXECUTOR_2:  %RMU−I−DATRECSTO,  3 data records stored.
EXECUTOR_1:  %RMU−I−DATRECSTO,  3 data records stored.

$ rmu/load/log/parallel=(buf=50,exec=2) −    ! hangs forever
    [.db]testdb  tab1  tab1.unl
%RMU−I−EXECUTORMAP, Executor EXECUTOR_1 (pid: 2F033130) will load storage
area AREA1.
%RMU−I−EXECUTORMAP, Executor EXECUTOR_2 (pid: 2F031731) will load storage
area AREA2.
```

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.6 RMU Extract Bugchecks when Extracting Some Modules

Bug 3911423

In Oracle Rdb Release 7.1.3, RMU Extract of a module could fail with an ACCVIO and bugcheck. This happened when the semicolon separating the SQL stored routine header from the routine body was followed by text such as a comment.

The following example shows this problem.

```
$ RMU/EXTRACT/ITEM=MODULE CPXM_DW_NO1/OUTP=NL:
%SYSTEM−F−ACCVIO, access violation, reason mask=00, virtual
address=0000000000000020, PC=FFFFFFFF80C84E00, PS=0000001B
%RMU−F−FATALOSI, Fatal error from the Operating System Interface.
%RMU−F−FTL_RMU, Fatal error for RMU operation at 24−SEP−2004 11:55:17.28
```

The bugcheck dump has a summary similar to:

- Alpha OpenVMS 7.3−1
- Oracle Rdb Server 7.1.3.0.1
- Got a RMUEXTBUGCHK.DMP
- SYSTEM−F−ACCVIO, access violation
- Exception occurred at symbol not found
- Called from DISPLAY_DESCRIPTION + 00001BA8
- Called from DISPLAY_SOURCE + 0000053C
- Called from EXTRACT_COLLATING_SEQ + 00008178
- Running image RMUEXTRACT71.EXE
- Dump created: 24−SEP−2004 11:55:17.22

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.7 Incorrect Backup of Empty Single AIJ File

Bug 3878051

Starting with Oracle Rdb Release 7.1.2.3, if you backed up an empty single extensible AIJ file, that is an AIJ with only an Open record, the "last commit TSN" field in the AIJ file open record was incorrectly reset to zero as shown below.

```
$ rmu/dump/after/nodata foo_aij.aij

1/1              TYPE=O, LENGTH=510, TAD=24-SEP-2004 04:48:43.32, CSM=00
    Database $111$DUA4:[VIGIER.RECO_BUG.DB]FOO.RDB;2
    Database timestamp is 24-SEP-2004 04:48:41.11
    Facility is "RDMSAIJ ", Version is 711.1
    Database version is 71.0
    AIJ Sequence Number is 0
    Last Commit TSN is 0:0
        :
```

Then when trying to recover using that AIJ file, it was failing with the recovery process ignoring all transactions from that AIJ file, as shown below.

```
$ rmu/recover foo_aij.aij
%RMU-I-LOGRECDB, recovering database file $111$DUA4:[VIGI.RECO_BUG.DB]FOO.RDB;1
%RMU-I-LOGOPNAIJ, opened journal file RAID1:[VIGI.RECO_BUG.DB]FOO_AIJ.AIJ;1
at 24-SEP-2004 05:07:56.35
%RMU-I-LOGRECSTAT, transaction with TSN 0:128 ignored
%RMU-I-AIJONEDONE, AIJ file sequence 0 roll-forward operations completed
%RMU-I-LOGRECOVR,  0 transactions committed
%RMU-I-LOGRECOVR,  0 transactions rolled back
%RMU-I-LOGRECOVR,  1 transaction ignored
        :
```

To avoid the problem, one can use multiple circular AIJs instead of a single extensible AIJ.

Now when backing up an empty single extensible AIJ file, no backup file is produced, and the AIJ open record is left unchanged. AIJ roll forward now works as expected.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.8 RMU/SHOW AFTER_JOURNAL/BACKUP_CONTEXT Reset RDM$AIJ_BACKUP_SEQNO Symbol to −1

Bug 3728347

In previous releases, RMU/SHOW AFTER_JOURNAL/BACKUP_CONTEXT reset RDM$AIJ_BACKUP_SEQNO symbol to −1 while RMU/BACKUP/AFTER was setting it correctly, as shown below.

```
$ rmu/backup/after/nolog db ""
$ sh symbol RDM$AIJ_BACKUP_SEQNO
  RDM$AIJ_BACKUP_SEQNO == "0"
$ rmu/show after/backup/out=tdb:x.x db
$ sh symbol RDM$AIJ_BACKUP_SEQNO
  RDM$AIJ_BACKUP_SEQNO == "-1"
```

Now RMU/SHOW AFTER_JOURNAL/BACKUP_CONTEXT correctly sets the RDM$AIJ_BACKUP_SEQNO symbol, as shown below.

```
$ rmu/backup/after/nolog db ""
$ sh symbol RDM$AIJ_BACKUP_SEQNO
  RDM$AIJ_BACKUP_SEQNO == "0"
$ rmu/show after/backup/out=tdb:x.x db
$ sh symbol RDM$AIJ_BACKUP_SEQNO
  RDM$AIJ_BACKUP_SEQNO == "0"
```

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.9 RMU/UNLOAD/DELETE_ROWS Deleted Unload Files on Error

Bug 3887611

The RMU/UNLOAD/DELETE_ROWS command, which deletes rows from an Rdb database table as they are unloaded, deleted the Unload file and the Record Definition file if one was being used, if an unrecoverable error occurred during the unload. This meant that data could be lost because the data already unloaded and deleted could not be reloaded using the Unload and/or the Record Definition files.

Now the UNLOAD file, and the RECORD DEFINITION file if it exists, will not be deleted if they contain any data when an error occurs that terminates the RMU/UNLOAD/DELETE_ROWS command. This will allow the data that has been deleted and stored in the UNLOAD file to be reloaded using the RMU/LOAD command.

Also if /COMMIT_EVERY is used with /DELETE_ROWS, the /COMMIT_EVERY value will have to be equal to or a multiple of the /ROW_COUNT value or a syntax error will be output.

```
$ RMU/UNLOAD/ROW_COUNT=5/COMMIT_EVERY=2/DELETE_ROWS MF_PERSONNEL -
_$ EMPLOYEES EMPLOYEES
%RMU-F-DELROWCOM, For DELETE_ROWS or FLUSH=ON_COMMIT the COMMIT_EVERY value must
 equal or be a multiple of the ROW_COUNT value.
The COMMIT_EVERY value of 2 is not equal to or a multiple of the ROW_COUNT value
 of 5.
%RMU-F-FTL_UNL, Fatal error for UNLOAD operation at 27-OCT-2004 08:55:14.06
```

This, along with other processing changes, will make sure that a database commit of the deleted rows will not take place until the rows have been successfully written to the UNLOAD file and any data left in internal RMS buffers has been flushed to the UNLOAD file. If an error occurs which aborts the unload, the deleted and unloaded rows can then be reloaded from the unload files. If an error occurs, the error handler will output the following message.

```
$ RMU/UNLOAD/DELETE_ROWS/ROW_COUNT=50/COMMIT_EVERY=50 MF_PERSONNEL -
 EMPLOYEES EMPLOYEES.UNL

%RMU-E-OUTFILNOTDEL, Fatal error, the output file is not deleted but may not
be useable,
50 records have been unloaded.
-COSI-F-WRITERR,  write error
-RMS-F-FUL, device full (insufficient space for allocation)
```

If a record definition file is not being used, the correct end codes will be written to the UNLOAD file before it is closed to avoid some of the UNLOADED data being rejected when the data is loaded using RMU/LOAD.

Note that Oracle cannot guarantee that all the deleted data is saved in the UNLOAD file or that the UNLOAD file will be able to reload all or some of the data using RMU/LOAD. RMU assumes that error handling can continue to perform flush, write and close operations on the UNLOAD file. If there is no virtual memory available or there are problems writing data, etc, some or all of the data may not be saved to the UNLOAD file or be reloadable. Where the problem occurs or the severity of the problem can determine how much data can be saved in the UNLOAD file or whether RMU/LOAD will have problems processing the UNLOAD file.

Qualifiers have also been added to RMU to allow flushing of the RMS buffers and retention of the UNLOAD files on error even if the /DELETE_ROWS qualifier is not used. These new qualifiers are documented in Section 4.1.3. Note that for /DELETE_ROWS, these new qualifiers do not have to be used unless the default behavior described above is not desired by the user. These qualifiers are /[NO]ERROR_DELETE, /FLUSH=BUFFER_END and /FLUSH=ON_COMMIT. If /DELETE_ROWS is specified, the defaults for these qualifiers are /NOERROR_DELETE and /FLUSH=ON_COMMIT. Otherwise the defaults are /ERROR_DELETE and to flush the RMS buffers only when the unload files are closed.

The following example shows that the output file is deleted when the RMU/UNLOAD error occurs even though rows have been deleted from the EMPLOYEES table after every 50 rows have been unloaded from the database. Therefore an RMU/LOAD command cannot be used to reload the data.

```
$ RMU/UNLOAD/DELETE_ROWS/ROW_COUNT=50/COMMIT_EVERY=50 MF_PERSONNEL -
 EMPLOYEES EMPLOYEES.UNL

%RMU-E-OUTFILDEL, Fatal error, output file deleted
-COSI-F-WRITERR,  write error
-RMS-F-FUL, device full (insufficient space for allocation)
```

The workaround for this problem is to use RMU/BACKUP to back up the database before doing the RMU/UNLOAD/DELETE_ROWS.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.3.10 RMU /SHOW LOCKS Limits Relaxed

Bug 3963053

Previously, the "/LOCK=" and "/PROCESS=" qualifiers of the "RMU /SHOW LOCKS" command were limited to 32 specified values.

This problem has been corrected in Oracle Rdb Release 7.1.4. The "/LOCK=" and "/PROCESS=" qualifiers of the "RMU /SHOW LOCKS" command now accept up to 256 values each.

2.3.11 RMU Extract May Output Illegal DECLARE FUNCTION Syntax

Bug 4060241

In prior releases of Oracle Rdb, the RMU Extract Item=Forward_References command would display the NOT DETERMINISTIC clause as part of a DECLARE FUNCTION. This clause is not required for DECLARE FUNCTION and a syntax error is reported when the script is executed.

The following example shows this error:

```
SQL> -- RMU/EXTRACT for Oracle Rdb V7.1-301             15-DEC-2004 13:48:06.79
SQL> --
SQL> --                          Routine Forward References
SQL> --
SQL> --------------------------------------------------------------------------
SQL> declare function MYABS (
cont>     in    :ARG
cont>         INTEGER)
cont>     returns
cont>         INTEGER
cont>     not deterministic
    not deterministic
    ^
%SQL-W-LOOK_FOR_STT, Syntax error, looking for:
%SQL-W-LOOK_FOR_CON,            (, BY, LANGUAGE, ;,
%SQL-F-LOOK_FOR_FIN,    found NOT instead
SQL> ;
;
^
%SQL-F-LOOK_FOR_STMT, Syntax error, looking for a valid SQL statement, found
; instead
SQL>
```

This problem has been corrected in Oracle Rdb Release 7.1.4. RMU Extract no longer generates this clause for DECLARE FUNCTION.

2.3.12 RMU/COLLECT STATISTICS=STORAGE Caused Incorrect Row Clustering Factors

RMU/COLLECT OPTIMIZER /STATISTICS=STORAGE could produce incorrect Row Clustering Factor values when used by itself without CARDINALITY statistics also being collected. This was because if only storage statistics were collected, the table cardinality values needed to calculate the row clustering factors were added to the previous table cardinality values. The table cardinality values should have been cleared before they were recalculated. This has been fixed.

The following example shows a case where collecting only storage statistics created row clustering factors which were half the correct value. The correct values were returned if both storage and cardinality values were collected. The correct row clustering factors would also have been produced if the /STATISTICS qualifier was not specified because it defaults to /STATISTICS=(CARDINALITY,WORKLOAD,STORAGE). The difference is due to the cardinality being in effect doubled since it was added to the previous cardinality which had not changed. Cardinality is used as a divisor in calculating the row cluster factor which caused the row cluster factors to be half the correct value if only STORAGE statistics were collected.

```
$ rmu/collect optimizer mf_personnel /statistics=(storage)/log=opt_stor.log
$ rmu/collect optimizer mf_personnel /statistics=(storage,cardinality)-
_$ /log=opt_storcard.log
$ search opt*.log "row clustering"


******************************

    DEVICE:[DIRECTORY]OPT_STORCARD.LOG;2
```

```
          Row clustering factor  : 0.0112994

          Row clustering factor  : 0.0114943

          Row clustering factor  : 1.0000000

          Row clustering factor  : 0.4285714

          Row clustering factor  : 0.0089286

          Row clustering factor  : 0.0180723

          Row clustering factor  : 0.0361446

          Row clustering factor  : 0.0131579

          Row clustering factor  : 0.0087464

          Row clustering factor  : 0.0000000


        ******************************

        DEVICE:[DIRECTORY]OPT_STOR.LOG;2


          Row clustering factor  : 0.0056497

          Row clustering factor  : 0.0057143

          Row clustering factor  : 0.5000000

          Row clustering factor  : 0.2142857

          Row clustering factor  : 0.0044643

          Row clustering factor  : 0.0090361

          Row clustering factor  : 0.0180723

          Row clustering factor  : 0.0065789

          Row clustering factor  : 0.0043732

          Row clustering factor  : 0.0000000
```

To workaround this problem, do not specify the collection of only STORAGE statistics but also collect CARDINALITY statistics. Do any of the following.

```
$ rmu/collect optimizer mf_personnel /statistics=(storage,cardinality)
$ rmu/collect optimizer mf_personnel −
_$ /statistics=(storage,cardinality,workload)
$ rmu/collect optimizer mf_personnel
```

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.4 LogMiner Errors Fixed

2.4.1 Various Logminer Failures After Upgrading to 7.1.3

Bugs 3949580 and 3992845

After upgrading to Oracle Rdb Release 7.1.3, the Logminer (RMU/EXTRACT/AFTER) facility would sometimes fail with various symptoms. For example, it might fail with the following error:

```
%RMU−F−FILACCERR, error writing work file DEV:[DIR]DFB3IMWS1G3.TMP;1
−SYSTEM−F−BADPARAM, bad parameter value
```

Also, the RMU process may fail with a bugcheck dump containing the following exceptions:

```
***** Exception at 00341DD4 : KUTREC$AIJBL_GET + 00000284
%COSI−F−BUGCHECK, internal consistency failure

***** Exception at 007174BC : KUTREC$AIJBL_GET_NEXT_BUFFER + 000000FC
%COSI−F−BUGCHECK, internal consistency failure
```

An internal error in the new filtering and buffer management algorithm would cause the buffer state to become incorrect, leading to various failure symptoms.

There is no workaround for this issue.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.4.2 RMU /UNLOAD /AFTER_JOURNAL /IGNORE OLD_VERSION Keyword

The RMU /UNLOAD /AFTER_JOURNAL command treats non−current record versions in the AIJ file as a fatal error condition. That is, attempting to extract a record that has a record version not the same as the table's current maximum version results in a fatal error.

There are, however, some very rare cases where a verb rollback of a modification of a record may result in an old version of a record being written to the after image journal even though the transaction did not actually complete a successful modification to the record. The RMU /UNLOAD /AFTER_JOURNAL command detects the old record version and aborts with a fatal error in this unlikely case.

The RMU /UNLOAD /AFTER_JOURNAL command now accepts a new keyword to partially work around this case. The /IGNORE qualifier has been enhanced to include the keyword OLD_VERSION. When this keyword is present, the RMU /UNLOAD /AFTER_JOURNAL command displays a warning message for each record that has a non−current record version and the record is not written to the output stream. The OLD_VERSION keyword accepts an optional list of table names to indicate that only the specified tables are permitted to have non−current record version errors ignored.

This problem has been corrected in Oracle Rdb Release 7.1.4.

2.5 Row Cache Errors Fixed

2.5.1 RCS Memory Leak When Snapshots in Cache Enabled

Bug 4079167

When using the "Snapshots in Row Cache" feature with prior releases of Oracle Rdb, the Row Cache Server (RCS) process leaked a small amount of virtual memory every 30 seconds. After a number of days, the RCS process could run out of memory and could fail with a "COSI−F−VASFULL, virtual address space full" status.

The logical name "RDM$BIND_RCS_SNAPSHOT_SWEEP_SECS" can be used to help reduce the rate of virtual memory consumption. The default behavior is for snapshot caches to be evaluated every 30 seconds to determine if any are "clogged" (or full of snapshot records that cannot be reclaimed). It is this timer that causes the memory leak. If the logical name is defined to a larger value (300, for example), memory will be lost at a slower rate. The disadvantage of a larger value, however, is that it will take longer to detect snapshot caches that have reclaimable space once they do become "clogged".

This problem has been corrected in Oracle Rdb Release 7.1.4. The RCS process no longer leaks this virtual memory.

Chapter 3
Software Errors Fixed in Oracle Rdb Release 7.1.3

This chapter describes software errors that are fixed by Oracle Rdb Release 7.1.3.

3.1 Software Errors Fixed That Apply to All Interfaces

3.1.1 Deadlock on Quiet After Journal Backup

Bug 3550383

Any process with multiple attaches to the same Rdb database in the same connection might experience a deadlock when a quiet point journal switch occurs while the process is starting a transaction. This deadlock is caused by contention for the quiet point lock. It will happen when the ABS process requests the quiet point lock after it has been acquired for the first attach associated with the transaction but before it is requested by the application process for the second attach. This is a "fatal embrace" situation and the application process operation will eventually fail with a deadlock returned.

For example, consider the following embedded SQL program statement for an application with two attaches to the same database (aliases DB1 and DB2):

```
EXEC SQL set transaction read write reserving
         db1.employees for protected write,
         db1.candidates for protected write,
         db1.colleges for protected write;
```

If the above SET TRANSACTION were executing at the instant a quiet point AIJ switch occurred, the application might fail and receive a return status of %RDB−E−DEADLOCK.

The problem has now been fixed so that the Rdb Dispatch layer will automatically detect this deadlock and retry. While it is still possible to get a deadlock, the condition described above will no longer cause one.

One possible workaround is to code the application to retry if the SET TRANSACTION fails due to a deadlock.

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.1.2 Logical Name Search Lists Supported by Attach

Bug 552106

If you defined a logical name that pointed to a search list of Rdb databases and if the leading databases were invalid in any way, an attach would fail to take place to the next one in the list. In the following example, if the database "disk1:[dir1]db1.rdb" does exist, SQL should attach to the next database "disk2:[dir2]db2.rdb".

```
$ define log1 disk1:[dir1]db1.rdb,disk2:[dir2]db2.rdb
$ SQL
SQL> attach 'filename log1';
```

However, instead of attaching to db2, SQL would give the following error:

```
%SQL−F−ERRATTDEC, Error attaching to database log1
−RDB−E−BAD_DB_FORMAT, log1 does not reference a database known to Rdb
−RMS−E−FNF, file not found
```

The problem has now been fixed so that SQL will correctly attach to the next database in the search list.

As a workaround, put a colon after the logical name in the attach expression. For the example above, one would use "log1:" instead of "log1".

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.1.3 RDB$CLIENT_DEFAULTS.DAT in SYS$LOGIN Incorrectly Used

Bug 852943

Defining SQL_DEFAULTS_RESTRICTION SYSTEM did not prevent the group and user RDB$CLIENT_DEFAULTS and RDB$SERVER_DEFAULTS from being used. When using a remote Rdb connection, there are a set of configuration files used to set various parameters for the client side of the remote connection. These are explained in the Oracle Rdb Installation and Configuration Guide and are processed by the remote client in a system−group−user hierarchy. If the logical RDB$SYSTEM_DEFAULTS is defined to point to a directory which contains a RDB$CLIENT_DEFAULTS.DAT file with the entry "SQL_DEFAULTS_RESTRICTION SYSTEM", it prevents the group and user−level RDB$CLIENT_DEFAULTS.DAT files (including the default one in SYS$LOGIN) from being processed. Likewise, a group−level RDB$CLIENT_DEFAULTS.DAT file pointed to by the logical RDB$GROUP_DEFAULTS in the group logical names table can prevent the user−level RDB$CLIENT_DEFAULTS.DAT file from being processed if it has the entry "SQL_DEFAULTS_RESTRICTION GROUP". Neither of these restrictions was being enforced.

There is an analogous hierarchy of RDB$SERVER_DEFAULTS files processed by RDBSERVER on the server side of a remote connection which uses the same mechanism ("SQL_DEFAULTS_RESTRICTION") to cut off processing of lower level defaults files. This was not being enforced either.

The problem has now been fixed so that the remote client and remote server will process the SQL_DEFAULTS_RESTRICTION correctly.

As a workaround, if no group or user−level defaults are desired, make sure that there is no RDB$GROUP_DEFAULTS or RDB$USER_DEFAULTS logical defined and that there is no RDB$CLIENT_DEFAULTS.DAT file in the SYS$LOGIN directory of the client account or the server account.

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.1.4 RDBPRE Incorrectly Reported an Error for Reserved Word VIA

Bug 3635081

In Oracle Rdb V7.1, the RDBPRE precompiler treated the name VIA as a reserved word which was a change in behavior from prior versions. Applications that used VIA as a field name would get a failure when compiling RDBPRE sources, as shown in the following example:

```
$  RCO PROVA
```

```
&RDB&        COD_VIA = B.VIA;
*                           ^
* *** ERROR, one of the following symbols was expected:
;          * name RDB$VALUE RDB$LENGTH RDB$DB_KEY
```

This problem has been corrected in Oracle Rdb Release 7.1.3. The VIA keyword is only reserved when following the FIRST keyword.

Note

The FIRST VIA ... FROM syntax can be used to execute subqueries from other relations (tables) and, unlike the FIRST ... FROM clause, will return a missing value instead of raising an exception.

3.1.5 Query With OR Predicate Slows Down Due to Wrong Strategy

Bug 3319289

The following query suffers in performance when the optimizer does not apply static OR index retrieval strategy:

```
set flags 'strategy,detail';
select count(*) from ( SELECT col33, col37 FROM MYTABLE1
            WHERE col34 = −463362512350317888 and
                ( ('A' = 'A' and
                    col10 >= '20031001' and
                    col10 <= '20031202' ) OR
                  ('B' = 'C' and
                    col09 >= '20031001' and
                    col09 <= '20031202' ) ) AND
                col20 >= 0 AND
                col20 <= 9999999999999 AND
                col13 >= ' ' AND
                col13 <= 'Z';
Tables:
  0 = MYTABLE1
Aggregate: 0:COUNT (*)
Merge of 1 entries
  Merge block entry 1
  Leaf#01 BgrOnly 0:MYTABLE1 Card=8483
    Bool: (0.COL34 = −463362512350317888) AND ((('A' = 'A') AND (0.COL10 >=
          '20031001') AND (0.COL10 <= '20031202')) OR (('B' = 'C') AND (0.COL09
           >= '20031001') AND (0.COL09 <= '20031202'))) AND (0.COL20 >= 0) AND (
          0.COL20 <= 9999999999999) AND (0.COL13 >= ' ') AND (0.COL13 <= 'Z')
    BgrNdx1 X2_MYTABLE [1:1] Fan=9
      Keys: 0.COL34 = −463362512350317888
    BgrNdx2 X7_MYTABLE [0:0] Fan=7
      Bool: (0.COL20 >= 0) AND (0.COL20 <= 9999999999999)

        8483
1 row selected
```

The same query runs much faster in Rdb Release 7.0.6.2, picking the right indices with static OR index retrieval:

```
Tables:
  0 = MYTABLE1
Aggregate: 0:COUNT (*)
Merge of 1 entries
  Merge block entry 1
  Conjunct: (0.COL34 = −463362512350317888) AND ((('A' = 'A') AND (0.COL10 >=
            '20031001') AND (0.COL10 <= '20031202')) OR (('B' = 'C') AND (
            0.COL09 >= '20031001') AND (0.COL09 <= '20031202'))) AND (0.COL20
             >= 0) AND (0.COL20 <= 9999999999999) AND (0.COL13 >= ' ') AND (
            0.COL13 <= 'Z')
  OR index retrieval
    Conjunct: ('A' = 'A') AND (0.COL10 >= '20031001') AND (0.COL10 <= '20031202'
              )
    Get     Retrieval by index of relation 0:MYTABLE1
      Index name  X2_MYTABLE [2:2]
        Keys: (0.COL34 = −463362512350317888) AND (0.COL10 >= '20031001') AND (
              0.COL10 <= '20031202')
        Bool: 'A' = 'A'
    Conjunct: NOT (('A' = 'A') AND (0.COL10 >= '20031001') AND (0.COL10 <=
              '20031202')) AND ('B' = 'C') AND (0.COL09 >= '20031001') AND (
              0.COL09 <= '20031202')
    Get     Retrieval by index of relation 0:MYTABLE1
      Index name  X3_MYTABLE [2:2]
        Keys: (0.COL34 = −463362512350317888) AND (0.COL09 >= '20031001') AND (
              0.COL09 <= '20031202')
        Bool: 'B' = 'C'
```

Notice that Rdb Release 7.0.6.2 applies static OR index retrieval while Rdb Release 7.1.2.4 uses regular index retrieval strategy.

As a workaround, if index X1_MYTABLE is removed, the query works the same as it does in Rdb Release 7.0.6.2 and it still works if the same index is created back again.

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.1.6 Bugcheck from Query with ORDER BY Using Translation Function

Bug 3733006

Queries using translate and "8−bit" characters could cause a bugcheck dump similar to:

```
***** Exception at 00C91AD4 : COSI_CS_TRANSLATE_TABLE + 000004F4
%COSI−F−BUGCHECK, internal consistency failure
```

The following example shows a typical query which could fail.

```
        SELECT
        EX.APEL1, EX.APEL2, EX.NOMBRE
        FROM
        EXPEDIENTES EX
        INNER JOIN MATRICULADOS M ON (M.ID_ALUMNO       = EX.ID_ALUMNO  AND
                                      M.AÑO             = 2004 AND
                                      M.CONVO           = 'J' )
        INNER JOIN TRIBUNALES T   ON (T.PLAN            = EX.PLAN        AND
                                      T.AÑO_SEL         = 2004 AND
                                      T.CONVO_SEL       = 'J' AND
                                      T.CENTRO          = EX.CENTRO)
        ORDER BY
                 TRANSLATE(EX.APEL1,'ÁÉÍÑÓÚÜ',
                                   'AEINOUU'),
                 TRANSLATE(EX.APEL2,'ÁÉÍÑÓÚÜ',
                                   'AEINOUU'),
                 EX.NOMBRE;
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.1.7 Bugcheck from a Count Query at Compile Time

Bug 3697894

The following simple query that counts the number of rows from the table bugchecks.

```
select count(*) from FA_STU_STAT
    WHERE (TIME_STAMP IS NOT NULL)
        AND (RC_ID = '0215')
        AND (TECH_ID =  '00067111')
        AND (AWD_YR = '2001')
        AND (TIME_STAMP > '6−FEB−2001 12:00');
%DEBUG−I−DYNMODSET, setting module RDMS$GEN_EXPR
%SYSTEM−F−ACCVIO, access violation, reason mask=00, virtual address=000
break on exception at RDMS$GEN_EXPR\RDMS$$CREATE_DESCRIPTOR\%LINE 4112
```

As a workaround, the query works if SQL flag 'nocount_scan' is set.

```
SQL> set flags 'nocount_scan';

Tables:
  0 = FA_STU_STAT
Aggregate: 0:COUNT (*)
Index only retrieval of relation 0:FA_STU_STAT
  Index name  FA_STU_STAT_INDEX [4:4]
    Keys: (0.RC_ID = '0215') AND (0.TECH_ID = '00067111') AND (0.AWD_YR = '2001'
          ) AND (0.TIME_STAMP > '6−FEB−2001 12:00:00.00') AND (NOT MISSING (
          0.TIME_STAMP))

           0
1 row selected
```

The query also works if the index FA_STU_STAT_INDEX is modified to use the sorted B−tree index, as in the following example.

```
create unique index FA_STU_STAT_INDEX on FA_STU_STAT (
    RC_ID, TECH_ID, AWD_YR, TIME_STAMP)    type is SORTED;
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.1.8 Logical Names Translated Twice

Starting with Oracle Rdb Release 7.1, many logical name translation requests within Rdb components were erroneously performed two times.

This problem has been corrected in Oracle Rdb Release 7.1.3. Logical names are no longer double translated.

3.1.9 AIP Length Not Set Correctly for Tables with AUTOMATIC Columns

In prior releases of Oracle Rdb V7.1, the record length stored in the AIP (area inventory pages) was incorrect if a table contained AUTOMATIC columns and did not have a storage map. That is, CREATE TABLE would not correctly calculate the length. This may lead to INSERT operations performing excessive I/O looking for free space.

A subsequent CREATE STORAGE MAP would use the correct length as would an ADD CACHE clause when adding a logical area row cache.

This problem was corrected in Oracle Rdb Release 7.1.2.

If any unmapped tables have this problem (i.e. were created with AUTOMATIC columns), they can be repaired using the RMU Repair statement.

```
$ RMU/REPAIR/INITIALIZE=LAREA_PARAMETERS:SYS$INPUT MF_PERSONNEL
NEW_EMPLOYEES/LENGTH=n
```

The value 'n' is the sum of all column lengths (add an additional 2 bytes for each VARCHAR column). Each row includes overhead for the row version number (2 bytes) and sufficient space for the null bit vector. Do not include any COMPUTED BY columns.

The following module and example can be used to verify the length setting in the AIP. Execute SQL$SAMPLE:INFO_TABLES.SQL to create the Rdb information tables in the database because the table RDB$LOGICAL_AREAS is required.

Note

ALTER TABLE ... ADD COLUMN, ALTER TABLE ... DROP COLUMN and ALTER TABLE ... ALTER COLUMN can also lead to similar errors in the AIP length. Smaller deviations are probably not significant for most applications.

```
create module SHOW_ROW_LENGTH_MODULE

function SHOW_ROW_LENGTH (in :rn rdb$object_name)
returns integer
comment is 'Query the system tables to calculate the'
/       'row length for the named table';
begin
declare :col_len, :max_field_id integer;
-- first count columns and record length of row
-- Note: computed by columns contribute nothing
-- but automatic columns do - use BITSTRING to differentiate
-- Note: VARCHAR type includes 2 byte length field
```

```
select sum(case
            when (f.rdb$computed_source is not null
                  and bitstring (f.rdb$flags from 2 for 2) = 0)
            then 0 -- computed by columns do not contribute
            else (case
                    when f.rdb$field_type = 37 -- VARCHAR type
                    then f.rdb$field_length + 2
                    else f.rdb$field_length
                  end)
           end),
           max (rf.rdb$field_id)
 into :col_len, :max_field_id
 from rdb$relations r, rdb$relation_fields rf, rdb$fields f
 where r.rdb$relation_name = :rn
    and rf.rdb$relation_name = r.rdb$relation_name
    and rf.rdb$field_source = f.rdb$field_name
    and r.rdb$view_blr is null;   -- exclude VIEWS

-- add in row overheads
-- We must include the size of the null bit vector which
-- is based on the maximum rdb$field_id
return :col_len
       + 5      -- header field
       + 2      -- version number
       + TRUNC ((:max_field_id + 7) / 8); -- null bit vector
end;

end module;
```

The following script uses the SHOW_ROW_LENGTH function to highlight tables which may need repair.

```
set flags 'noprefix,trace';

begin
declare :display_header integer = 1;
declare :mismatch_count integer = 0;

-- display all user tables
-- ignore views
for :r as select rdb$relation_name
          from rdb$relations
          where rdb$system_flag = 0
            and rdb$view_blr is null
          order by rdb$relation_name
do
    begin
    declare :stored_val, :actual_val integer;
    if (:display_header = 1)
    then
        trace cast('Table Name' as rdb$object_name),
              cast('Actual' as char(11)),
              cast('AIP Length' as char(11));
        trace cast('----------' as rdb$object_name),
              cast('------' as char(11)),
              cast('----------' as char(11));
        set :display_header = 0;
    end if;
    set :stored_val =
          (select rdb$record_length
           from rdb$logical_areas
           where rdb$logical_area_name = :r.rdb$relation_name
           limit to 1 row);
    set :actual_val =
          SHOW_ROW_LENGTH (:r.rdb$relation_name);
    trace :r.rdb$relation_name,
          :actual_val,
          :stored_val,
          (case when :stored_val <> :actual_val
                then '<****'
                else ''
           end);
    if :stored_val <> :actual_val
    then
        set :mismatch_count = :mismatch_count + 1;
    end if;
    end;
end for;

if :mismatch_count <> 0
then
    trace '';
    trace 'The tables marked with <**** should be examined';
end if;
end;
```

Some example output is shown here. Note the <**** highlighting a table which has a smaller than expected AIP length.

```
$ sql$ @rel
Table Name                 Actual     AIP Length
----------                 ------     ----------
A100                       12         12
A101                       14         14
A102                       14         14
A103                       14         14
A104                       22         16          <****
A200                       12         12
A201                       14         14
A202                       14         14
A203                       14         14
A204                       16         16

The tables marked with <**** should be examined more closely
```

3.1.10 Bugcheck Dump During Alter Storage Map Command

Bug 2825363

Under extremely rare circumstances, it was possible that an *ALTER STORAGE MAP* command could cause various bugchecks due to memory corruption. The alter storage map must be causing rows to be moved, and an index must exist that has duplicate values and is of *TYPE IS SORTED RANKED*.

In the following example, the storage map is altered to move the rows in the table to a new set of storage areas. A sorted ranked index exists with many duplicate values so this index needs to be updated to reflect the new location of each record.

```
SQL> alter storage map DATA_MAP
```

```
cont>  partitioning is not updatable
cont>  store using (MY_ID)
cont>  in DATA_AREA_1  (threshold is (83)) with limit of (1)
cont>  in DATA_AREA_2  (threshold is (83)) with limit of (2)
cont>  in DATA_AREA_3  (threshold is (83)) with limit of (3)
cont>  otherwise in  DATA_AREA;
%RDMS−I−BUGCHKDMP, generating bugcheck dump file
MBRADLEY_USR:[BRADLEY]RDSBUGCHK.DMP;
%RDMS−I−BUGCHKDMP, generating bugcheck dump file
MBRADLEY_USR:[BRADLEY]SQLBUGCHK.DMP;
```

The exception in the bugcheck will be an access violation, and will usually be in one of the memory management routines such as *COSI_MEM_GET_VM* or *COSI_MEM_FREE_VM*.

There are three ways to avoid the problem:

- The offending index can be dropped prior to altering the storage map and recreated afterwards.
- The table can be unloaded and truncated prior to altering the storage map. The data can then be sucessfully loaded when the alter completes.
- If the index is rebuilt with a fullness threshold less than 100%, the problem should not occur.

The problem is extremely rare and does not cause corruption to the data or index. If the problem occurs, the database is automatically recovered and one of the methods described above can be used to avoid the problem.

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.1.11 Complex Query With MAX()...GROUP BY Returns Wrong Result

Bug 3719771

The customer states that a complex nested SQL query with a MAX()...GROUP BY clause returns the wrong result (1 row selected) when the cardinality of the table is increased to 2.1 million rows. If the SQL flag 'MAX_STABILITY' is enabled, the query returns the correct result which is 65 rows out of 2094 rows.

After simplifying the query by removing non−significant parts and deleting the unneeded data rows (down to 3 rows), we come out with the following example query where the main select returns columns from the two main sub−selects, Q2 and Q5.

The problem occurs in the Q5 sub−select clause. Note that the sub−select Q5 itself is made up of two nested sub−selects, Q67 and Q89 which are nearly identical sub−queries except for the WHERE clauses. The query returns the incorrect result if the Q67 columns are selected for Q5, but it returns the correct result if the Q89 columns are selected instead.

```
create data file test_bug;

create table TAB1 (
C01_ID                 CHAR(11)       ,
C05_ID                 CHAR(1)        ,
C02_LOC                CHAR(2)        ,
OPERATION_NO           CHAR(5)        ,
FINISH_DATIME          TIMESTAMP(2)   ,
C02_PRODUCT_ID         CHAR(3)        ,
C08_CODE                 CHAR(1)        ,
C06_COUNT                INTEGER        ,
C07_COUNT                INTEGER
);

INSERT INTO TAB1 (C01_ID, C05_ID, C02_LOC, OPERATION_NO, FINISH_DATIME,
    C02_PRODUCT_ID, C08_CODE, C06_COUNT, C07_COUNT) VALUE
  ('1K46804424', '2', '20', '51000', timestamp'2004−06−01 20:17:31.00',
   'DAB', '0', 1, 0);

INSERT INTO TAB1 (C01_ID, C05_ID, C02_LOC, OPERATION_NO, FINISH_DATIME,
    C02_PRODUCT_ID, C08_CODE, C06_COUNT, C07_COUNT) VALUE
  ('1K46804420', '2', '20', '51000', timestamp'2004−06−01 20:15:38.00',
   'DAB', '0', 1, 0);

INSERT INTO TAB1 (C01_ID, C05_ID, C02_LOC, OPERATION_NO, FINISH_DATIME,
    C02_PRODUCT_ID, C08_CODE, C06_COUNT, C07_COUNT) VALUE
  ('1K46804417', '2', '20', '51000', timestamp'2004−06−01 20:16:28.00',
   'DAB', '0', 1, 0);

create unique index TAB1_IDX_S1 on TAB1 (
    C01_ID, C02_LOC, OPERATION_NO, FINISH_DATIME);

create unique index TAB1_IDX_S3 on TAB1 (
    OPERATION_NO, C01_ID, C05_ID, C02_LOC, FINISH_DATIME);

create unique index TAB1_IDX_S4 on TAB1 (
    FINISH_DATIME, C01_ID, C05_ID, C02_LOC, OPERATION_NO);

update RDB$RELATIONS
             set RDB$CARDINALITY=2100000  where RDB$RELATION_NAME='TAB1';
commit;
disconnect all;

attach file 'test_bug';
set flags 'strategy';
select
    Q2.C01_ID,
    Q2.C02_LOC,
    Q5.C01_ID
  from
     (select
          Q3.C01_ID,
          Q3.C02_LOC,
          Q3.OPERATION_NO,
          T1.C07_COUNT
      from
         (select
              Q4.C01_ID,
              Q4.C02_LOC,
              Q4.OPERATION_NO,
              Q4.C07_COUNT
          from
             (select
                  C01_ID,
                  C02_LOC,
                  OPERATION_NO,
                  C07_COUNT
              from TAB2
                 where
                    OPERATION_NO between '50000' and '60000' and
                    FINISH_DATIME between timestamp'2004-06-01 20:15:38.00'
                      and timestamp'2004-06-01 20:17:31.00'
                        ) as Q4
                ) as Q3,
              TAB1 as T1
         where
             Q3.C01_ID = T1.C01_ID
             and Q3.C02_LOC = T1.C02_LOC
             and Q3.OPERATION_NO = T1.OPERATION_NO
         ) as Q2,
     (select
          ! the query returns the wrong result if Q67 is referenced here
          !
          Q67.C01_ID,          ! Q89.C01_ID,
          Q67.C02_LOC,         ! Q89.C02_LOC,
          Q67.OPERATION_NO,    ! Q89.OPERATION_NO,
          Q67.C07_COUNT        ! Q89.C07_COUNT
      from
         (select
              C01_ID,
              C02_LOC,
              OPERATION_NO,
              max(C06_COUNT) as C06_COUNT,
              C07_COUNT
          from TAB1
              where
                  OPERATION_NO between '50000' and '60000' AND
                  FINISH_DATIME between timestamp'2004-06-01 20:15:38.00'
                    and timestamp'2004-06-01 20:17:31.00'
              group by
                  C01_ID,
                  C02_LOC,
                  OPERATION_NO,
                  C07_COUNT
                ) as Q67,
             (select
                  C01_ID,
                  C02_LOC,
                  OPERATION_NO,
                  max(C06_COUNT) as C06_COUNT,
                  C07_COUNT
              from TAB1
                  where
                      FINISH_DATIME between timestamp'2004-06-01 20:15:38.00'
                        and timestamp'2004-06-01 20:17:31.00'
                  group by
                      C01_ID,
                      C02_LOC,
                      OPERATION_NO,
                      C07_COUNT
                ) as Q89
        where
            Q67.C01_ID = Q89.C01_ID and
            Q67.C02_LOC = Q89.C02_LOC and
            Q67.OPERATION_NO = Q89.OPERATION_NO
        ) as Q5
where
    Q2.C01_ID = Q5.C01_ID and
    Q2.C02_LOC = Q5.C02_LOC and
    Q2.OPERATION_NO = Q5.OPERATION_NO
;
```

```
 Q2.C01_ID     Q2.C02_LOC   Q5.C01_ID
 1K46804417    20           1K46804417
1 row selected
```

As a workaround, the query returns the correct result if the select statement of Q5 is changed to reference Q89 instead of Q67, as in the following example.

```
select
    Q2.C01_ID,
    Q2.C02_LOC,
    Q5.C01_ID
  from
     (select
          Q3.C01_ID,
          Q3.C02_LOC,
          Q3.OPERATION_NO,
          T1.C07_COUNT
      from
         (select
              Q4.C01_ID,
              Q4.C02_LOC,
              Q4.OPERATION_NO,
              Q4.C07_COUNT
          from
             (select
                  C01_ID,
                  C02_LOC,
                  OPERATION_NO,
                  C07_COUNT
              from TAB2
                 where
                    OPERATION_NO between '50000' and '60000' and
                    FINISH_DATIME between timestamp'2004−06−01 20:15:38.00'
                      and timestamp'2004−06−01 20:17:31.00'
                        ) as Q4
                ) as Q3,
              TAB1 as T1
         where
             Q3.C01_ID = T1.C01_ID
             and Q3.C02_LOC = T1.C02_LOC
             and Q3.OPERATION_NO = T1.OPERATION_NO
         ) as Q2,
     (select
          ! the query returns the correct result if Q89 is referenced here
          !
          Q89.C01_ID,
          Q89.C02_LOC,
          Q89.OPERATION_NO,
          Q89.C07_COUNT
      from
         (select
              C01_ID,
              C02_LOC,
              OPERATION_NO,
              max(C06_COUNT) as C06_COUNT,
              C07_COUNT
          from TAB1
              where
                  OPERATION_NO between '50000' and '60000' AND
                  FINISH_DATIME between timestamp'2004−06−01 20:15:38.00'
```

```
                     and timestamp'2004-06-01 20:17:31.00'
               group by
                   C01_ID,
                   C02_LOC,
                   OPERATION_NO,
                   C07_COUNT
                 ) as Q67,
              (select
                   C01_ID,
                   C02_LOC,
                   OPERATION_NO,
                   max(C06_COUNT) as C06_COUNT,
                   C07_COUNT
               from TAB1
                   where
                       FINISH_DATIME between timestamp'2004-06-01 20:15:38.00'
                         and timestamp'2004-06-01 20:17:31.00'
                   group by
                       C01_ID,
                       C02_LOC,
                       OPERATION_NO,
                       C07_COUNT
                 ) as Q89
        where
            Q67.C01_ID = Q89.C01_ID and
            Q67.C02_LOC = Q89.C02_LOC and
            Q67.OPERATION_NO = Q89.OPERATION_NO
        ) as Q5
where
    Q2.C01_ID = Q5.C01_ID and
    Q2.C02_LOC = Q5.C02_LOC and
    Q2.OPERATION_NO = Q5.OPERATION_NO
;
 Q2.C01_ID     Q2.C02_LOC   Q5.C01_ID
 1K46804417    20           1K46804417
 1K46804420    20           1K46804420
 1K46804424    20           1K46804424
3 rows selected
```

The key parts of this query which contributed to the situation leading to the error are these:

1. The query joins four contexts of the same table TAB1.
2. The strategy is a cross join with 2 entries (e.g. Q2 and Q5) where the first entry (Q2) is a merge of a match strategy (Q3 and T1) with a sort node, and the second entry (Q5) is another cross join of 2 entries (Q67 and Q89), as in the following example.

```
Cross block of 2 entries
  Cross block entry 1                <== representing Q2
    Merge of 1 entries
      Merge block entry 1
      Conjunct
      Match
        Outer loop                   <== representing  Q3 and Q4
          Sort                       <== this sort node is another key part
          Merge of 1 entries
            Merge block entry 1
            Merge of 1 entries
              Merge block entry 1
              Conjunct        Conjunct
              Leaf#01 BgrOnly TAB1 Card=2100000
```

```
                    BgrNdx1 TAB2_IDX_S3 [1:1] Bool Fan=8
                    BgrNdx2 TAB2_IDX_S4 [1:1] Bool Fan=8
                Inner loop      (zig−zag)    <== representing  T1
                  Conjunct        Conjunct        Conjunct        Conjunct        Get
                  Retrieval by index of relation TAB1
                    Index name  TAB1_IDX_S1 [0:0]
          Cross block entry 2                <== representing  Q5
            Merge of 1 entries
              Merge block entry 1
              Cross block of 2 entries
                Cross block entry 1          <== representing  Q89
                  Merge of 1 entries
                    Merge block entry 1
                    Aggregate       Sort    Conjunct        Conjunct
                    Leaf#02 BgrOnly TAB1 Card=2100000
                      BgrNdx1 TAB1_IDX_S4 [1:1] Fan=8
                      BgrNdx2 TAB1_IDX_S3 [2:2] Bool Fan=8
                Cross block entry 2          <== representing  Q67
                  Merge of 1 entries
                    Merge block entry 1
                    Aggregate       Sort    Conjunct        Conjunct
                    Leaf#03 BgrOnly TAB1 Card=2100000
                      BgrNdx1 TAB1_IDX_S1 [4:4] Bool Fan=9
```

3. The sub−select queries under the query Q5 contains a GROUP BY clause with MAX aggregate function.
4. The index TAB1_IDX_S3 is used to retrieve the outer leg of the cross and also used in the inner leg of the cross.
5. Three of the columns referenced by the SELECT statements are defined in TAB1_IDX_S3 as C01_ID, C02_LOC and OPERATION_NO.
6. The column FINISH_DATIME referenced by the BETWEEN clause is also defined in TAB1_IDX_S3.
7. The SELECT statement of the query Q5 references the columns of the query Q67 which becomes the inner leg of the cross strategy, instead of the outer leg as explicitly suggested by the SQL query.

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.1.12 Unexpected Constraint Failures from RMU Verify Constraint

Bugs 2303741, 2649462 and 3449121

In prior releases of Oracle Rdb, it was possible in rare cases to see one of the following commands report a failure while validating referential integrity constraints. Repeating the action was often successful.

- RMU/VERIFY/CONSTRAINT
- ALTER TABLE ... ENABLE CONSTRAINTS
- TRUNCATE TABLE

This problem was caused by a timing condition in the table verify code. The following example shows one of the errors.

```
SQL>Alter table degrees enable all constraints;
%RDMS−I−BUGCHKDMP, generating bugcheck dump file DISK1:[TEST]RDSBUGCHK.DMP;
```

The bugcheck summary looks similar to this:

- Alpha OpenVMS 7.3−2
- Oracle Rdb Server X7.1−00
- Got a RDSBUGCHK.DMP
- SYSTEM−F−ILLEGAL_SHADOW, illegal formed trap shadow, Imask=05AC5DCC, Fmask=0000001B, summary=C0, PC=0000000000000000, PS=00000000
- Exception occurred at symbol not found
- Database root: DISK1:[TEST.CONSTRAINTS]MF_PERSONNEL_SQL

RMU Verify Constraints may simply report a constraint failure (RMU−I−CONSTFAIL, Verification of constraint "<name>" has failed). Defining the logical name RDMS$SET_FLAGS and assigning the value ITEM_LIST will cause RMU to report the failure status of the constraint verification. This may include the following:

```
~H: ...verify constraint "SOMETABLE_SOMECOLUMN_NOT_NULL4"
~H: ...verify complete with failure status 000005B4

Status 000005B4 is "%SYSTEM−F−ILLEGAL_SHADOW, illegal formed trap shadow,
Imask=<hex>, Fmask=<hex>,  summary=!XB, PC=<hex>, PS=<hex>".

~H: ...verify constraint "THISTABLE_THATCOLUMN_NOT_NULL10"
~H: ...verify complete with failure status 0000000C

Status 0000000C is "SYSTEM−F−ACCVIO, access violation, reason mask=!XB,
virtual address=<hex>,  PC=<hex>, PS=<hex>".
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.2 SQL Errors Fixed

3.2.1 High I/O Rates for BUILD PARTITION on HASHED Index

Bug 3500867

The BUILD or REBUILD PARTITION clauses of ALTER INDEX can be used to build just one partition of an index. In the case of SORTED indices, the keys are sorted in the index defined order prior to creating the index partition. This ensures optimal buffer use during the build.

However, in prior releases of Oracle Rdb V7.1, there was no similar optimization done for HASHED indices. This problem report showed that the random distribution of the generated target pages might cause pages to be flushed from the buffers and later re−read when new keys mapped to that page. Thus the reported I/O was much greater than expected and might exceed the total I/O for a full CREATE INDEX statement.

A revised algorithm is now used to read the key values for the HASHED index and then sort them on the predicted target page. Now when the HASHED index partition is built, the operations work on the current buffers eliminating or greatly reducing the chance of page re−read. Table 3−1, Comparison of BUILD PARTITION Changes shows the I/O comparison for the reported database. Results for other indices will vary according to key distribution, number of rows and hardware being used.

Table 3−1 Comparison of BUILD PARTITION Changes

Using V7.1.2.4		
CREATE INDEX	about 145000 DIO	26 seconds cpu
ALTER INDEX ... BUILD PARTITION (1/3 of the index)	about 458000 DIO	40 seconds cpu
Using V7.1.3		
CREATE INDEX	about 146000 DIO	24 seconds cpu
ALTER INDEX ... BUILD PARTITION (1/3 of the index)	about 50000 DIO	7 seconds cpu

Increasing the NUMBER OF BUFFERS could be used as a workaround for the reported problem, allowing pages to remain in memory longer.

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.2.2 Unexpected LANUNSDTP Error for BIGINT Parameters

Bug 3609753

The SQL Module Language compiler would erroneously report that the calling language did not support BIGINT.

The following example shows the reported warning.

```
    :large_val          BIGINT
                        1
%SQL−W−LANUNSDTP, (1) FORTRAN does not support the data type for parameter
LARGE_VAL
```

This problem has been corrected in Oracle Rdb Release 7.1.3. The current releases of the C, BASIC, FORTRAN and Pascal compilers each support a 64 bit (aka BIGINT) data type. Therefore, this warning is no longer required.

3.2.3 Simple Function Names Hiding User Column Data

Bugs 2319321 and 3649003

In prior releases of Oracle Rdb, simple function names occluded similarly named columns in a table. In recent versions, Rdb has added several builtin functions that do not require a parameter list and applications that have successfully used these names as column names now receive the function result instead of the column value.

If a column name is unqualified (by table name or correlation name) or not delimited, it may be interpreted as an SQL builtin function that does not require parameter lists (known as simple functions). Such column names would be: CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_DATE, LOCALTIME, LOCALTIMESTAMP, SYSDATE, ROWNUM, CURRENT_USER, SESSION_USER, SYSTEM_USER, USER, CURRENT_UID, SESSION_UID, SYSTEM_UID and UID.

The following example shows that UID returns unexpected results (in this case a different data type).

```
SQL> select * from t1;
 F1        UID
 A          Joe
 B          Tim
2 rows selected
SQL> select uid from t1;
     4653080
     4653080
2 rows selected
```

When the SELECT * FROM ... statement is used, the column value is returned.

This problem has been corrected in Oracle Rdb Release 7.1.3. These simple functions are now treated as transparent in SQL. That is, they will only be used as simple functions if there is no visible column in the current query scope with that name. If the column is delimited or qualified by a table, view or correlation name, then it will also be unambiguous.

If the function result is desired then you can use an alternate format for the function (e.g. adding the fractional precision as in CURRENT_TIME(2), using an equivalent function CURRENT_UID instead of UID, or writing a SQL function which has its own query scope).

SQL now also issues a warning message describing the change in behavior.

```
$ SQL$MOD APP
 select user, current_time into :a, :b from t2t;
        1
%SQL−W−ABMCOLNAME, (1) Column name matches simple function USER; column value
```

```
used
 select user, current_time into :a, :b from t2t;
              2
%SQL−W−ABMCOLNAME, (2) Column name matches simple function CURRENT_TIME; column
value used
```

3.2.4 SQL$MOD Compile Bugcheck at GEM_IP_BUILD + 00003F7C

Bug 1921858

If a SQL Module Language procedure declared a variable which was initialized with an expression involving one of the procedure's parameters, the SQL$MOD compile would fail and produce a SQLBUGCHK.DMP.

For example, consider the following SQL Module Language program. In the procedure "FORMAT_INPUT", the variable ":MY_RESULT" is initialized to the value of the parameter ":IN_STRING".

```
MODULE                  BUG
DIALECT                SQL92
LANGUAGE                COBOL
AUTHORIZATION          RDB$DBHANDLE
PARAMETER COLONS
QUOTING RULES          SQL92
−−
DECLARE ALIAS FILENAME PERSONNEL
−−
PROCEDURE FORMAT_INPUT
        ( SQLSTATE,
          :IN_STRING        CHAR(255),
          :OUT_STRING       CHAR(255));

begin atomic
declare :MY_RESULT char(255) = 'The input was:' || :IN_STRING;
set :OUT_STRING = :MY_RESULT;
end;
```

When the above program was compiled, SQL$MOD would fail and produce a SQLBUGCHK.DMP file which contained entries similar to the following:

```
***** Exception at 0015E92C : GEM_IP_BUILD + 00003F7C
%SYSTEM−F−ACCVIO, access violation, reason mask=00, virtual address=00000000000
```

The problem has now been fixed so that SQL$MOD will correctly compile the program.

As a workaround, code the procedure to use an assignment statement instead of initialization in the declaration. Refering to the example above, recode the declaration of :MY_RESULT as follows:

```
declare :MY_RESULT char(255);
set :MY_RESULT = 'The input was:' || :IN_STRING;
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.2.5 Unexpected Bugcheck when Using TRACE Statement and Subselect

Bug 3013618

In prior releases of Oracle Rdb, it was possible to receive a bugcheck dump when displaying a subselect with the TRACE statement. For example, the following TRACE statement bugchecks when SET FLAGS 'TRACE' is enabled.

```
SQL> set flags 'trace';
SQL> set noexecute
SQL>
SQL> begin -- block
cont> declare :AGENT_CODE integer = 0;
cont>
cont> trace
cont>     (SELECT CHAIN_CODE
cont>      FROM AGENTS
cont>      WHERE GROUP_CODE IN ('JVH','JV0')
cont>        AND AGENT_CODE = :agent_code
cont>      LIMIT TO 1 ROWS),
cont>     '|';
cont>
cont> end; -- block
%RDMS-I-BUGCHKDMP, generating bugcheck dump file USER2:[TESTING]RDSBUGCHK.DMP;
%SQL-I-BUGCHKDMP, generating bugcheck dump file USER2:[TESTING]SQLBUGCHK.DMP;
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
virtual address=0000000000000000, PC=000000000024A4EC, PS=0000001B
```

A workaround for this problem is to assign the subselect result to a local variable and then use that local variable with the TRACE statement.

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.2.6 DEFAULT on COMPUTED BY Causes Corruption During INSERT

Bug 3711193

In rare circumstances, it is possible that COMPUTED BY columns are given DEFAULT values (possibly by some third party tool). This is not possible to do directly with SQL, as shown below, where both CREATE TABLE and ALTER TABLE restrict the DEFAULT clause.

```
SQL> create table t (a integer, c computed by a*2 default 0);
%SQL-F-DEFVALNOTCB, Default values are not allowed for COMPUTED BY or
AUTOMATIC INSERT columns
SQL> create table t (a integer);
SQL> alter table t add column c computed by a*2 default 0;
%SQL-F-DEFVALNOTCB, Default values are not allowed for COMPUTED BY or
AUTOMATIC INSERT columns
SQL> alter table t add column c computed by a*2;
SQL> alter table t alter column c default 0;
%SQL-F-DEFVALNOTCB, Default values are not allowed for COMPUTED BY or
AUTOMATIC INSERT columns
```

The Rdb Server made assumptions about what type of columns could include DEFAULT values and incorrectly applied the default value to the row, overwriting the leading columns of the row.

This problem has been corrected in Oracle Rdb Release 7.1.3. Oracle Rdb now ignores any DEFAULT defined for a COMPUTED BY column and no longer applies them during INSERT.

3.2.7 Unexpected NO_META_UPDATE Error Generated by DROP MODULE ... CASCADE When Attached by PATHNAME

Bug 755182

In prior releases of Oracle Rdb, operations such as DROP MODULE ... CASCADE would sometimes generate an unexpected NO_META_UPDATE error. This occurred when the session attached to a database by PATHNAME. For example:

```
SQL> drop module m1 cascade;
%RDB−E−NO_META_UPDATE, metadata update failed
−RDMS−F−OBJ_INUSE, object "M1P1" is referenced by M2.M2P1 (usage: Procedure)
−RDMS−E−MODNOTDEL, module "M1" has not been deleted
```

Oracle CDD/Repository does not support CASCADE for this object type and so this clause was ignored. The workaround is to attach by FILENAME and perform the metadata operation.

In this release of Oracle Rdb, an informational message is issued by Interactive SQL describing the downgrade from CASCADE to RESTRICT in such cases. This is demonstrated by the following example:

```
SQL> create module m language sql procedure ttt (); trace 's'; end module;
%CDD−I−IGNOREINFO, unsupported entity − omitted at mblr offset 1
SQL> drop module m cascade;
%SQL−I−DWNGRDCLS, Option CASCADE is not supported by this version
and has been downgraded to RESTRICT
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.2.8 Relaxed Rules for Date/Time Comparison

Bug 3016466

In prior releases of Oracle Rdb, comparison operations which mixed DATE ANSI, DATE VMS and TIMESTAMP types were required to use the CAST function to derive a common type for comparison. With this release of Rdb, these rules have been relaxed. Expressions can now freely mix DATE (ANSI or VMS) and TIMESTAMP types without requiring the CAST function. In addition, differing fractional second precision for TIMESTAMP types is no longer checked by SQL.

The comparison operators are:

- equals (=)
- not equals (<>)
- less than (<)
- less than or equals (<=)
- greater than (>)

- greater than or equals (>=)
- and BETWEEN

INTERVAL and TIME types still require that compatible types be used. However, subtype, leading field precision, and fractional second precision need not be equal.

The following example shows the prior behavior where a SQL error is generated.

```
SQL> select count(employee_id)
cont> from salary_history
cont> where salary_start between date'1980−1−1' and current_timestamp(2);
%SQL−F−UNSDATXPR, Unsupported date expression
−SQL−F−DATEEQLILL, Operands of date/time comparison are incompatible
SQL>
```

This is the new behavior from SQL:

```
SQL> select count(employee_id)
cont> from salary_history
cont> where salary_start between date'1980−1−1' and current_timestamp(2);

        1035
1 row selected
SQL>
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.2.9 Unexpected ONEDBINMOD Error Reported for LOCK TABLE Statement

Bug 3725189

In prior releases of Oracle Rdb V7.1, the LOCK TABLE could not be used in a compound statement when more than one database was attached. The following examples show the reported error:

```
SQL> -- legal
SQL>  begin on alias a
cont>  lock table employees for shared read mode;
cont>  end;
%SQL−F−ONEDBINMOD, Only one alias is legal in this module
SQL>
SQL> −− legal
SQL>  begin on alias a
cont>  lock table a.employees for shared read mode;
cont>  end;
%SQL−F−ONEDBINMOD, Only one alias is legal in this module
```

The use of the ON ALIAS restricts the compound statement to just one of the databases. This problem has been corrected in Oracle Rdb Release 7.1.3. LOCK TABLE is now supported when using the ON ALIAS clause.

3.2.10 Unexpected Memory Error when CONSTRAINT=ON Option Used for an Application

Bugs 3700378 and 3836073

In prior releases of Oracle Rdb V7.1.2, long running transactions within SQL module language or SQL precompiled applications that were compiled using the CONSTRAINT=ON option could fail with errors due to insufficient virtual memory. A summary of a bugcheck dump due to this problem is shown below.

- Alpha OpenVMS 7.3−1
- Oracle Rdb Server 7.1.2.2.1
- Got a RDSBUGCHK.DMP
- COSI−F−VASFULL, virtual address space full
- SYSTEM−F−ILLPAGCNT, illegal page count parameter
- Exception occurred at COSI_MEM_GET_VM + 000005E4
- Called from COSI_MEM_GET_POOL + 00000048
- Called from RDMS$$TOP_RESERVING_LIST + 00000864
- Called from BLI$CALLG + 000000BC

This is caused by a new control block being allocated for each statement executed in the transaction. In this case, millions of statements were executed during a single transaction.

Applications that receive this error should be analyzed to see if the /SQLOPTION=CONSTRAINT=ON (or IMMEDIATE) or /CONSTRAINT=ON (or IMMEDIATE) are required for the application. A workaround is to remove these qualifiers from the compile of all modules in the application and rebuild.

This problem has been corrected in Oracle Rdb Release 7.1.3. This control block is no longer allocated by Oracle Rdb.

3.2.11 Changes to Pascal Support in SQL Precompiler

As part of work to modernize the SQL precompiler for Pascal, changes have been made to the SQL defined types to make use of Pascal language data types.

The following types are now declared as INTEGER64 and are no longer defined as quadword sized RECORD structures.

- SQL_BIGINT
- SQL_QUADWORD

If your application requires access to the sub fields (L0 and L1) of the definition, then SQL$QUADWORD should be used since it continues to be defined as a record structure.

The following types are now declared as UNSIGNED64 and are no longer defined as quadword sized RECORD structures.

- SQL_DATE_ANSI
- SQL_DATE
- SQL$DATE

The following types are now declared as INTEGER16 and are no longer defined as numeric ranges.

- SQL_INDICATOR
- SQL_SMALLINT
- SQL$INDICATOR
- SQL$SMALLINT

The following types are now declared as INTEGER8 and are no longer defined as numeric ranges.

- SQL_BYTE
- SQL$BYTE

This change should eliminate occassional warnings from Pascal such as that shown in the following example.

```
        var_quad    := date_msg  ;
.......................^
%PASCAL−W−EMPTYVAR, Fetching an empty record with an explicit size attribute
may not yield expected results at line number 217 in file
USER2:[TESTING.WORK.PASCAL]XX.PAS;6
```

The variables DATE_MSG and VAR_QUAD in this example are both declared using the type SQL$DATE.

3.2.12 New Optional Builtin Function RDB$$IS_ROW_FRAGMENTED

Bug 3788472

This release of Oracle Rdb supports a new optional builtin function that can determine if a row is fragmented. The function, RDB$$IS_ROW_FRAGMENTED, must be declared as function using the attributes and properties as shown below.

```
declare function RDB$$IS_ROW_FRAGMENTED
    (in :dbk char(8) character set unspecified)
    returns integer;
```

The following example shows the usage on the WORK_STATUS table in the PERSONNEL database.

```
SQL> declare function RDB$$IS_ROW_FRAGMENTED
cont>    (in :dbk char(8) character set unspecified)
cont>    returns integer;
SQL>
SQL> select dbkey, RDB$$IS_ROW_FRAGMENTED (dbkey) from work_status;
                  DBKEY
              99:10:12            0
              99:10:13            0
              99:10:14            0
3 rows selected
```

Usage Notes

- A function result of zero (0) indicates a non−fragmented row and a value of one (1) indicates a fragmented row.
- This routine may only be used from Interactive and Dynamic SQL.

- Only valid DBKEY values should be passed to the function.
- If the DBKEY passed is not the current row then additional I/O may be required to fetch the target row.
- If the DBKEY is for a vertically partitioned table then only the fragmented state of the primary segment is reported. There is currently no programatic method to determine fragmented secondary segments.
- Temporary table and information table rows are never fragmented as they reside in virtual memory only. Tables stored in a row cache will never be fragmented.
- Fragmentation occurs when either the row is too large to fit entirely on a page or an existing row was updated to a larger size and no space existed at that time for the expanded row. The first case requires that the page size be changed for the area. However, for the second case a DELETE and INSERT of the row might remove the fragmentation. In that case, this function allows the DBA to identify candidate fragmented rows. Fragmentation may occur when compression is enabled and the compressed row size changes due to changed data, NULL values replaced with non−NULL values, or ALTER TABLE or ALTER DOMAIN statements that have increased the size of columns.

3.2.13 ROUND and TRUNC are now Built In Functions for SQL

Bugs 1035809, 1158334, and 3385103

In prior versions of Oracle Rdb, the functions ROUND and TRUNC for numeric values were implemented as special functions in the SQL_FUNCTIONS library provided with Rdb V7.0 and later versions.

The implementation of these functions used the data type DOUBLE PRECISION for both parameter and results. This caused unexpected results due to the imprecise nature of floating point arithmetic. A value such as 4.185 would not round to 4.19 as expected because the internal (and approximate) representation of the number was really 4.184999942780E+000 and therefore did not appear to require rounding according to the rounding rules.

The following example shows this problem.

```
SQL> select cast(round (4.185,2) as integer(2)) from rdb$database;

          4.18
1 row selected
SQL> select cast(round (4.175,2) as integer(2)) from rdb$database;

          4.18
1 row selected
SQL>
```

The functions ROUND and TRUNC for numeric values are now supported as native functions in Oracle Rdb Release 7.1.3. If you use SQL to access older versions of Rdb (such as via remote access), then SQL will revert to the pre−V7.1.3 behavior and use the SQL functions provided by the SQL_FUNCTIONS library.

This problem has been corrected in Oracle Rdb Release 7.1.3. Fixed point values are now truncated and rounded correctly. Floating values, while supported by ROUND and TRUNC, may not always return the expected results. Please review usage of ROUND in such contexts.

The implementation of ROUND and TRUNC for DATE values requires the use of the OCI Services for Rdb library (also known as SQL*net for Rdb). This remains true with this release of Rdb, however, these functions will now accept DATE ANSI, TIMESTAMP and DATE VMS values.

Attempts to use ROUND or TRUNC on a database that is not set up for OCI Services will receive errors similar to these:

```
SQL> select TRUNC (current_date) from rdb$database;
%RDB−E−OBSOLETE_METADA, request references metadata objects that no longer exist
−RDMS−F−BAD_SYM, unknown routine symbol − TRUN2
SQL> select ROUND (current_date) from rdb$database;
%RDB−E−OBSOLETE_METADA, request references metadata objects that no longer exist
−RDMS−F−BAD_SYM, unknown routine symbol − ROUN2
```

3.3 RMU Errors Fixed

3.3.1 RMU Extract Item=VERIFY Incorrectly Includes Information Tables

Bug 3693672

In prior versions of Oracle Rdb V7.1, the RMU Extract Item=VERIFY option would include the optional INFORMATION tables in the list of logical areas to verify. However, these tables do not reside on disk and are materialized with data from within the Rdb root (.rdb) file or from Rdb server internal data.

When applying the generated script to verify the database, these error messages are displayed:

```
%RMU−F−NOTLAREA,  "RDB$CACHES" is not a logical area name
%RMU−F−NOTLAREA,  "RDB$CHARACTER_SETS" is not a logical area name
%RMU−F−NOTLAREA,  "RDB$DATABASE_JOURNAL" is not a logical area name
%RMU−F−NOTLAREA,  "RDB$DATABASE_ROOT" is not a logical area name
%RMU−F−NOTLAREA,  "RDB$DATABASE_USERS" is not a logical area name
%RMU−F−NOTLAREA,  "RDB$JOURNALS" is not a logical area name
%RMU−F−NOTLAREA,  "RDB$LOGICAL_AREAS" is not a logical area name
%RMU−F−NOTLAREA,  "RDB$NLS_CHARACTER_SETS" is not a logical area name
%RMU−F−NOTLAREA,  "RDB$STORAGE_AREAS" is not a logical area name
```

A workaround is to edit the script to remove the information tables. For example, the DCL SEARCH command can be used to filter the result generated by RMU Extract.

```
$ RMU/EXTRACT/ITEM=VERIFY SQL$DATABASE /OUTPUT=verify_file.com
$ SEARCH verify_file.com −
 "RDB$CACHES","RDB$CHARACTER_SETS","RDB$DATABASE_JOURNAL", −
 "RDB$DATABASE_ROOT","RDB$DATABASE_USERS","RDB$JOURNALS", −
 "RDB$LOGICAL_AREAS", "RDB$NLS_CHARACTER_SETS", −
 "RDB$STORAGE_AREAS" −
 /MATCH=NOR /OUTPUT=new_verify_file.com
```

This problem has been corrected in Oracle Rdb Release 7.1.3. Information tables are now excluded by RMU Extract Item=VERIFY.

3.3.2 RMU/BACKUP Did Not Always Create an RMUBUGCHK.DMP File for Certain Errors

Bug 3668497

For certain unexpected fatal errors that occurred at certain phases during an Oracle Rdb RMU database backup, the error was output but an RMUBUGCHCK.DMP file was not created. These errors included SS$_ACCVIO and similar unexpected statuses, including COSI$_ACCVIO, COSI$_BUGCHECK and COSI$_UNEXPERR. This problem was caused by the fatal error status being checked for not being available at the time the RMU exception handler, which was supposed to create the dump file, was called. This problem has been fixed and now an RMUBUGCHK.DMP file will be created in these cases.

The following example shows that if the unexpected SS$_ACCVIO VMS system access violation was signaled during an RMU/BACKUP of a database, an RMUBUGCHK.DMP file was not always created by RMU and therefore no message was output by RMU that referenced the file specification of the RMUBUGCHK.DMP file.

```
$RMU/BACKUP MF_PERSONNEL MFP.RBF
%SYSTEM−F−ACCVIO, access violation, reason mask=00, virtual
address=0000000000CC2000, PC=00000000003C4290, PS=0000001B
%RMU−F−FATALERR, fatal error on BACKUP
%RMU−F−FTL_BCK, Fatal  error for BACKUP operation at  1−JUN−2004 11:41:31.23
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.3.3 Default Concurrency for RMU/RESTORE from Multiple Tape Drives was Incorrect

Bug 3683557

For the Oracle Rdb RMU restore of a database from multiple tape drives under Oracle Rdb RMU V6.1 and Oracle Rdb RMU V7.0, the restore by default reads the tape drives sequentially (master/slave) unless the /VOLUMES qualifier is specified which causes the drives to be read concurrently (master/master).

However, because of a code change under Oracle Rdb RMU V7.1, the default was unintentionally changed so that the multiple tape drives were read concurrently (master/master). Note that this refers to the default case for reading tape drives where qualifiers such as /MASTER and /VOLUMES are not specified to indicate how tape drives are to be processed. This caused a problem where an unnecessary prompt for readying an extra unneeded tape volume to be read was output by RMU. This problem has been fixed and now the default for multiple tape drive processing for the Oracle Rdb V7.1 RMU/RESTORE command is sequential reading of the tape drives (master/slave), the same default as for previous versions of RMU.

The following example shows that before this problem was fixed an unnecessary prompt was output by RMU/RESTORE asking for a third tape volume to be readied on the first tape drive.

```
$ rmu/restore  −
_$  /direct=device:[directory]/nocdd/log  −
_$  $tapedrive1:mfp.rbf,$tapedrive2:  −
_$  /label=(label1,label2)
%RMU−I−RESTXT_00, Restored root file device:[directory]MF_PERSONNEL.RDB;1
%RMU−I−RESUME, resuming operation on volume 2 using _$tapedrive2
%RMU−I−RESTXT_21, Starting full restore of storage area (RDB$SYSTEM)
 device:[directory]MF_PERS_DEFAULT.RDA;1 at 22−MAY−2004 00:31:38.53
%RMU−I−RESTXT_24, Completed full restore of storage area (RDB$SYSTEM)
 device:[directory]MF_PERS_DEFAULT.RDA;1 at 22−MAY−2004 00:31:38.80
%RMU−I−RESUME, resuming operation on volume 3 using _$tapedrive1
%MOUNT−F−MEDOFL, medium is offline
%RMU−I−READYREAD, mount volume 3 label LABEL03 on _$tapedrive1: for reading
Press return when ready:
```

The workaround for this problem is to specify the /VOLUMES qualifier on the RMU/RESTORE command line.

```
$ rmu/restore  −
_$  /direct=device:[directory]/nocdd/log/volumes=2  −
_$  $tapedrive1:mfp.rbf,$tapedrive2:  −
```

```
_$  /label=(label1,label2)
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.3.4 RMU Extract may Bugcheck if Executed when Metadata Changes are Being Made

Bug 3684674

In previous releases of Oracle Rdb, it was possible that RMU Extract could bugcheck if other sessions were modifying the Rdb system table via commands such as DROP or ALTER. For example:

```
$ RMU/EXTRACT/OUT=DSM DSM
%RDMS−I−BUGCHKDMP, generating bugcheck dump file DUMMY:[DUMMY]RDSBUGCHK.DMP;
%RDB−F−BUG_CHECK, internal consistency check failed
%RMU−F−FATALRDB, Fatal error while accessing Oracle Rdb.
%RMU−F−FTL_RMU, Fatal error for RMU operation at 10−JUN−2004 14:20:58.06
```

The bugcheck summary would appear similar to this example.

- Alpha OpenVMS 7.2−1
- Oracle Rdb Server 7.0.7.1
- Got a RDSBUGCHK.DMP
- COSI−F−BUGCHECK, internal consistency failure
- Exception occurred at DIOBND$FETCH_AIP_ENT + 000001D4
- Called from DIOBND$GET_LACB + 00000144
- Called from RDMS$TOP_DATABASE_INFO + 00001D24
- Called from BLI$CALLG + 000000BC
- Running image RMUEXTRACT70.EXE

This problem occurs because the DROP operation has marked the logical area used by a table, storage map or index as deleted. The delete of the logical area happens immediate and is visible therefore to both read−only and read−write users. When RMU Extract requests information for the now deleted logical area, a bugcheck occurs noting the "internal consistency" problem.

In this release of Rdb, the bugcheck has been replaced with a more informative message, as shown below.

```
SQL> create database filename dsm
cont> create storage area rdb$system filename dsm
cont> create storage area a1 filename a1;
SQL> create table t1(i1 integer);
SQL> create storage map t1m for t1 store in a1;
SQL> commit;
SQL>  $ RMU/EXTRACT/OUTPUT=DSM DSM
SQL> drop storage map t1m;
SQL>  $ RMU/EXTRACT/OUTPUT=DSM DSM
%RDB−E−OBSOLETE_METADA, request references metadata objects that no longer exist
−RDMS−F−CANTFINDLAREA, cannot locate logical area 47 in area inventory page list
%RMU−F−FATALRDB, Fatal error while accessing Oracle Rdb.
%RMU−F−FTL_RMU, Fatal error for RMU operation at 10−JUN−2004 11:46:52.36
SQL>
```

This problem has been corrected in Oracle Rdb Release 7.1.3. If this error occurs, then repeat the RMU Extract command as this will start a new transaction.

3.3.5 RMU Unload Could Not be Used to Unload Information Tables

Bug 3692176

In prior versions of Oracle Rdb, it was not possible to RMU Unload from an Rdb system table or any table or view marked as a system table. In particular, the special INFORMATION tables in Rdb V7.1 contain information which might be useful to unload for analysis.

The following example shows the error reported by RMU Unload in prior versions.

```
$ RMU/UNLOAD SQL$DATABASE RDB$CACHES CURRENT_CACHE/REC=(FILE=RC,FORMAT=DELIM)
%RMU−E−OUTFILDEL, Fatal error, output file deleted
−RMU−F−RELNOTFND, Relation (RDB$CACHES) not found
```

This restriction has been lifted in Oracle Rdb Release 7.1.3. Now any table in the database can be unloaded using RMU Unload. However, RMU Load will not allow system tables to be the target of a load operation. Such action would corrupt the database metadata and leave the database in an unusable state.

3.3.6 RMU/RESTORE/INCREMENTAL/AREA Did Not Check if Default and List Areas Were Out of Date

Bug 3676698

On an incremental "by area" restore, where the specified list of storage areas to be restored does not include RDB$SYSTEM, a check is done to see if the backup RBF file contains the RDB$SYSTEM area. If so, and the incremental backup TSN of the live RDB$SYSTEM area is less than the TSN of the RDB$SYSTEM area in the RBF backup file (the live RDB$SYSTEM area is out of date), RMU requires that the newer RDB$SYSTEM area in the backup RBF file be added to the list of areas to be restored. In this case, RMU returns the error message:

```
%RMU−F−RDBSYSTEMREQ, The RDB$SYSTEM storage area must also be specified
```

This is to prevent database corruption.

However, if a LIST (SEGMENT) and/or DEFAULT storage area other than RDB$SYSTEM were defined for the database, no check was made for these areas either. Now these areas are checked for being out of date on an incremental "by area" RMU database restore the same way that the RDB$SYSTEM storage area is checked. The RDBSYSTEMREQ message has been changed to include one, two or all three of these areas as necessary.

```
%RMU−F−RDBSYSTEMREQ, The RDB$SYSTEM MF_PERS_DEFAULT MF_PERS_SEGSTR storage
 area(s) must also be specified for the restore.
```

The following example shows that previously only an out of date RDB$SYSTEM storage area was checked for on an incremental "by area" restore.

```
$rmu/restore/incremental/noconfirm/nocdd/nolog/area jobs.rbf JOBS
%RMU−F−RDBSYSTEMREQ, The RDB$SYSTEM storage area must also be specified
%RMU−F−FTL_RSTR, Fatal error for RESTORE operation at 14−JUL−2004 16:17:49.33
```

The following example shows that now the default and list (segment) areas will also be checked to make sure they are not out of date, in addition to the RDB$SYSTEM storage area, on an incremental "by area" restore.

```
$rmu/restore/incremental/noconfirm/nocdd/nolog/area jobs.rbf JOBS
%RMU−F−RDBSYSTEMREQ, The RDB$SYSTEM MF_PERS_DEFAULT MF_PERS_SEGSTR storage
 area(s) must also be specified for the restore.
%RMU−F−FTL_RSTR, Fatal error for RESTORE operation at 14−JUL−2004 16:17:49.33
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.3.7 RMU Extract Not Preserving BEFORE/AFTER COLUMN Ordering

Bug 2658043

In prior releases of Oracle Rdb, RMU Extract would extract the table using the order defined after an ALTER COLUMN ... BEFORE COLUMN or AFTER COLUMN clause was applied. Unfortunately, dependencies between columns are formed by the original CREATE TABLE and so it was possible to get a table definition which was not accepted by Rdb.

The following example shows such a problem:

```
$ rmu/extract/item=table/option=(noheader,match="T %") abc
set verify;
set language ENGLISH;
set default date format 'SQL92';
set quoting rules 'SQL92';
set date format DATE 001, TIME 001;
attach 'filename DISK2:[TESTING]ABC.RDB';
create table T (
    X
        computed by (A * 2),
    A
        INTEGER);

commit work;

$
$ sql$
SQL> attach 'filename abc';
SQL> drop table T;
SQL> commit;
SQL> exit;
$
$ sql$
SQL> set verify;
SQL> set language ENGLISH;
SQL> set default date format 'SQL92';
SQL> set quoting rules 'SQL92';
SQL> set date format DATE 001, TIME 001;
SQL> attach 'filename DISK2:[TESTING]ABC.RDB';
SQL> create table T (
cont>     X
cont>         computed by (A * 2),
cont>     A
cont>         INTEGER);
%SQL−F−FLDNOTCRS, Column A was not found in the tables in current scope
SQL>
SQL> commit work;
```

This problem has been corrected in Oracle Rdb Release 7.1.3. RMU Extract now generates an ALTER TABLE statement to reorder the columns as required.

3.3.8 RMU/DUMP/BACKUP/LIBRARIAN=READER=1 Could Create Multiple Reader Threads

The RMU/LIBRARIAN command can be used with RMU commands that access tape devices if an application that implements the Oracle Media Management API V2 and that supports Oracle Rdb is running. RMU/DUMP/BACKUP/LIBRARIAN=READER=1 dumps RBF files stored in the Media Management application archive. The "READER=1" parameter specifies that one RMU reader thread is to be used to dump one or multiple RBF files stored in the Media Management application archive that represent one database backup. READER=1 is the default.

There was a problem where READER=1 was ignored and the number of reader threads used by RMU was set equal to the number of RBF files stored in the Media Management application archive for one database backup. Therefore, if the backup files MFP.RBF, MFP.RBF02 and MFP.RBF03 were stored in the Media Management application archive and READER=1 was specified, three reader threads were created to do the dump, one for each file belonging to the database backup. So, although READER=1 was specified, RMU/DUMP/BACKUP/LIBRARIAN ignored it and incorrectly assumed the user had specified READER=3.

The following example shows that if three backup files were stored in the Media Management application archive for one backup file set named MFP and READER=1 were specified for the RMU/DUMP/BACKUP/LIBRARIAN command, three reader threads were created by RMU to do the dump when one thread should have been created to read the three files, one after the other. Note that thread creation is internal so it cannot be actually shown in this example. Now, READER=1 (the default) will not be ignored.

```
$ RMU/LIBRARY/LIST MFP

             LIBRARIAN BACKUP FILES

BACKUP NAME: MFP.RBF
   CREATION METHOD:      stream
   CREATION DATE/TIME:   Fri Jul 23 09:23:52 2004
   VOLUME LABEL:         VOL533
   SHARING MODE:         multiple users
   ORDERING MODE:        random access
   COMMENT:              Oracle Media Manger API
BACKUP NAME: MFP.RBF02
   CREATION METHOD:      stream
   CREATION DATE/TIME:   Fri Jul 23 09:23:53 2004
   VOLUME LABEL:         VOL533
   SHARING MODE:         multiple users
   ORDERING MODE:        random access
   COMMENT:              Oracle Media Manager API
BACKUP NAME: MFP.RBF03
   CREATION METHOD:      stream
   CREATION DATE/TIME:   Fri Jul 23 09:23:53 2004
   VOLUME LABEL:          VOL533
   SHARING MODE:          multiple users
   ORDERING MODE:         random access
   COMMENT:               Oracle Media Manager API
RYEROX>RMU/DUMP/BACKUP/LIBRARIAN=READER=1 MFP
%RMU−I−DMPTXT_163, No dump option selected. Performing read check.
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.3.9 RMU/BACKUP/LIBRARIAN=(LOGICAL=(logical_name=equivalence_na Behavior Changed

The RMU/LIBRARIAN command can be used with RMU commands that access tape devices if an application that implements the Oracle Media Management API V2 and that supports Oracle Rdb is running. One or more logicals can be defined as parameters to the /LIBRARIAN qualifier. These logicals are defined in the process table in user mode so they only exist while the RMU/LIBRARIAN command is executing. For example:

```
RMU/BACKUP/LIBRARIAN=(LOGICAL=(logical_name=equivalence_name)) MF_PERSONNEL MFP
```

This defines one or more process logicals and then stores the database backup "MFP" in the Oracle Media Management application archive. These logicals depend on the Media Management application being used and are documented in the Media Management application documentation. They might define the archive to store the backup in or a debug logical that is used by the particular Media Management application. These process logicals were being defined by RMU at backup or restore time when they should have been defined earlier (at the time the particular Media Management application was initialized). This could cause the Media Management application to ignore these logicals. This has now been corrected.

The following example shows an RMU backup of an Rdb database to a Media Management application where two logicals are defined that can be used by the Media Manager application.

```
RMU/BACKUP/LIBRARIAN=(LOGICAL=(logical1=equivalence1,logical2=equivalence2)) −
 MF_PERSONNEL MFP
```

This problem has been corrected in Oracle Rdb Release 7.1.3.

3.3.10 RMU Extract Fails on Some View Definitions

Bugs 1267126, 1729528, 2130670, 3294003, 3418796, 3518990 and 3810178

In prior releases of Rdb, RMU Extract failed to correctly extract view definitions when the view contained nested UNION, EXCEPT, MINUS, INTERSECT operators, derived tables or other complexities. Incorrect syntax would be generated for these cases.

This release also corrects a memory management problem which may cause RMU Extract to bugcheck.

These problems have been corrected in Oracle Rdb Release 7.1.3.

3.3.11 RMU Extract Fails to Extract Large Trigger Definition

Bug 3807411

In prior releases of Oracle Rdb, it was possible that RMU Extract would fail to extract a trigger definition with many large trigger actions. The string used to hold the formatted source is limited by the VMS string descriptor length of 65535 bytes.

The following example shows the error.

```
$ RMU/EXTRACT/ITEM=(TRIGGER) SQL$DATABASE
%STR−F−STRTOOLON, string is too long (greater than 65535)
%RMU−F−FATALOSI, Fatal error from the Operating System Interface.
%RMU−F−FTL_RMU, Fatal error for RMU operation at 29−JUL−2004 14:28:44.72
```

This problem has been corrected in Oracle Rdb Release 7.1.3. RMU Extract now displays the partial trigger at the end of each action to prevent this error occurring. However, if trigger definitions include actions that format to a string greater than 65535 bytes, then this error may appear again. Oracle recommends using SQL functions to reduce the complexity of these triggers if possible.

3.3.12 RMU Extract of a Module May Generate Incorrect Syntax

Bug 3867167

In previous versions of Oracle Rdb V7.1, the syntax generated by RMU Extract Item=Module may generate incorrect syntax for a procedure or function.

The following example shows an excerpt from an RMU Extract generated script. The highlighted text is incorrectly included in the output.

```
   .
   .
   .
create module M1
    language SQL

    function M1_F1 (
        )
        returns
            INTEGER
        not deterministic
        comment is
          'a function comment ; with a semicolon';
 with a semicolon'
variant
;
return 0;

end module;
commit work;
   .
   .
   .
```

This occurs when a COMMENT IS clause or DEFAULT clause in the original routine definition includes a semicolon (;). RMU Extract extracts the original SQL source for the routine body and uses the semicolon as the starting point.

This problem has been corrected in Oracle Rdb Release 7.1.3. A semicolon within a string literal is now ignored when looking for the start of the routine body.

3.4 LogMiner Errors Fixed

3.4.1 RMU /UNLOAD /AFTER_JOURNAL Possible Missing Pre−delete Content

Bug 3569886

In very rare cases, it was possible for the RMU /UNLOAD /AFTER_JOURNAL command to omit deleted record contents from an extract operation. The LogMiner was incorrectly processing the after−image journal when a record's pre−delete content was followed in the journal by a physical record delete indicator for the same physical DBKEY.

This problem has been corrected in Oracle Rdb Release 7.1.3. The RMU /UNLOAD /AFTER_JOURNAL command now uses a modified sort comparison phase that correctly handles the unexpected order in the after−image journal so that the pre−delete record content is extracted as expected.

Chapter 4
Enhancements Provided in Oracle Rdb Release 7.1.4

4.1 Enhancements Provided in Oracle Rdb Release 7.1.4

4.1.1 Support for OpenVMS Version 8.2

This version of Rdb, Release 7.1.4, supports HP's OpenVMS Version 8.2 Release.

4.1.2 RMU/BACKUP/PARALLEL/DISK_FILE Now Supports ALLOCATION_QUANTITY and EXTEND_QUANTITY

The RMU/BACKUP/PARALLEL/DISK_FILE command, which creates multiple executor processes to back up an Oracle Rdb database to multiple disk RBF backup files, now supports the ALLOCATION_QUANTITY and EXTEND_QUANTITY qualifiers which set the number of blocks for the allocation of each disk backup RBF file and the number of blocks for the extension of each disk backup RBF file. These values have been added to the RMU Backup PLAN file and will be applied to each backup RBF file created by a single parallel database backup command.

The following new syntax is now supported in the RMU parallel backup to disk PLAN file:

```
    Allocation_Quantity = 20480
    Extend_Quantity = 2048
```

If these values are not specified, the defaults for these values will be used.

The following example shows first the RMU/BACKUP/PARALLEL command which creates the PLAN file, then the RMU/BACKUP/PLAN command which executes the PLAN file, then a VMS TYPE command that shows the contents of the PLAN file with the entries for ALLOCATION_QUANTITY and EXTEND_QUANTITY.

```
$  RMU/BACKUP/log -
    /parallel=(exec=4)-
    /LIST_PLAN=DEVICE:[DIRECTORY]rdbbackup.plan-
              /ONLINE -
              /NOEXEC -
              /LOCK_TIMEOUT=7200 -
              /DISK_FILE=(writer_threads=4) -
              /CRC=CHECKSUM -
              /ACTIVE_IO=5 -
              /BLOCK=65024 -
              /ALLOCATION_QUANTITY=20480 -
              /EXTEND_QUANTITY=2048 -
              MF_PERSONNEL -
        DEVICE:[DIRECTORY]RDB_DB1.RBF, -
        DEVICE:[DIRECTORY], -
        DEVICE:[DIRECTORY], -
        DEVICE:[DIRECTORY]
$ RMU/BACKUP/PLAN rdbbackup.plan
$ TYPE rdbbackup.plan
 Plan created on 24-SEP-2004 by RMU/BACKUP.

Plan Name = RDBBACKUP
Plan Type = BACKUP


Plan Parameters:
    Database Root File = DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1
    Backup File = RDB_DB1.RBF
    Style = Multifile
    FixedDisks
    PromptAutomatic
    Active_IO = 5
    Reader_Thread_Ratio = 5
    Block_Size = 65024
    Lock_Timeout = 7200
    Allocation_Quantity = 20480
    Extend_Quantity = 2048
    Checksum_Verification
    CRC = Checksum
    NoIncremental
    Log
    Online
    Quiet_Point
    NoCompression
    NoRewind
    Statistics
    ACL
End Plan Parameters

Executor Parameters :
    Executor Name = COORDINATOR
    Executor Type = Coordinator
End Executor Parameters

Executor Parameters :
    Executor Name = WORKER_001
    Executor Type = Worker
    Start Storage Area List
        MF_PERS_SEGSTR,
        EMPIDS_OVER,
        RDB$SYSTEM
    End Storage Area List
    Writer_threads = 1
    Directory List
        DEVICE:[DIRECTORY]
    End Directory List
End Executor Parameters

Executor Parameters :
    Executor Name = WORKER_002
    Executor Type = Worker
    Start Storage Area List
        DEPARTMENTS,
        EMP_INFO
    End Storage Area List
    Writer_threads = 1
    Directory List
        DEVICE:[DIRECTORY]
    End Directory List
End Executor Parameters

Executor Parameters :
    Executor Name = WORKER_003
    Executor Type = Worker
    Start Storage Area List
        EMPIDS_LOW,
        JOBS
    End Storage Area List
    Writer_threads = 1
    Directory List
        DEVICE:[DIRECTORY]
    End Directory List
End Executor Parameters

Executor Parameters :
    Executor Name = WORKER_004
    Executor Type = Worker
    Start Storage Area List
        EMPIDS_MID,
        SALARY_HISTORY
    End Storage Area List
    Writer_threads = 1
    Directory List
        DEVICE:[DIRECTORY]
    End Directory List
End Executor Parameters
```

4.1.3 RMU/UNLOAD Now Supports /FLUSH and /[NO]ERROR_DELETE Qualifiers

The RMU/UNLOAD command by default deletes the Unload and Record Definition files if an unrecoverable error occurs which causes an abnormal termination of the unload command execution. The /[NO]ERROR_DELETE qualifier has now been added to RMU/UNLOAD to allow specifying whether or not the Unload and Record Definition files should be deleted on error.

The RMU/UNLOAD command by default flushes any data left in the internal RMS file buffers only when the Unload file is closed. A new /FLUSH=BUFFER_END qualifier has been added to flush the internal RMS buffers to the Unload file after each RMU/UNLOAD buffer has been written to the Unload file. A new /FLUSH=ON_COMMIT qualifier has been added to flush the internal RMS buffers to the Unload file just before the current RMU/UNLOAD transaction is committed. More frequent flushing of the internal RMS buffers will avoid the possible loss of some Unload file data if an error occurs and the Unload file is not to be deleted on error. Note that additional flushing of the RMS internal buffers to the Unload file can cause the RMU/UNLOAD to take longer to complete.

If /DELETE_ROWS is specified, the defaults for these qualifiers are /NOERROR_DELETE and /FLUSH=ON_COMMIT. This is to allow the Unload and Record Definition files to be used to reload the data if an unrecoverable error has occurred after the delete of some of the unloaded rows has been commited. Otherwise the defaults are /ERROR_DELETE and to flush the RMS buffers only when the unload files are closed. Note that even if the Unload file is retained, it may not be able to reload the data using RMU/LOAD if the error is severe enough to prevent the RMU error handler from continuing to access the Unload file once the error is detected.

The following example shows that if /FLUSH=ON_COMMIT is specified, the /COMMIT_EVERY value must be equal to or a multiple of the /ROW_COUNT value so the commits of unload transactions occur after the internal RMS buffers are flushed to the Unload file. This prevents loss of data if an error occurs.

```
$ RMU/UNLOAD/ROW_COUNT=5/COMMIT_EVERY=2/FLUSH=ON_COMMIT MF_PERSONNEL -
_$ EMPLOYEES EMPLOYEES
%RMU-F-DELROWCOM, For DELETE_ROWS or FLUSH=ON_COMMIT the COMMIT_EVERY value must
 equal or be a multiple of the ROW_COUNT value.
The COMMIT_EVERY value of 2 is not equal to or a multiple of the ROW_COUNT value
 of 5.
%RMU-F-FTL_UNL, Fatal error for UNLOAD operation at 27-OCT-2004 08:55:14.06
```

The following examples show that the Unload file and Record Definition files are not deleted on error if /NOERROR_DELETE is specified and that these files are deleted on error if /ERROR_DELETE is specified. Note that if the Unload file is empty when the error occurs, it will be deleted.

```
$ RMU/UNLOAD/NOERROR_DELETE/ROW_COUNT=50/COMMIT_EVERY=50 MF_PERSONNEL -
 EMPLOYEES EMPLOYEES.UNL

%RMU-E-OUTFILNOTDEL, Fatal error, the output file is not deleted but may not
be useable,
50 records have been unloaded.
-COSI-F-WRITERR,  write error
-RMS-F-FUL, device full (insufficient space for allocation)

$ RMU/UNLOAD/ERROR_DELETE/ROW_COUNT=50/COMMIT_EVERY=50 MF_PERSONNEL -
 EMPLOYEES EMPLOYEES.UNL

%RMU-E-OUTFILDEL, Fatal error, output file deleted
-COSI-F-WRITERR,  write error
-RMS-F-FUL, device full (insufficient space for allocation)
```

4.1.4 Altering COMPUTED BY and AUTOMATIC AS Columns

Prior releases of Oracle Rdb did not allow COMPUTED BY or AUTOMATIC AS columns to be altered. Specifically, the computed expression was set when the column was defined. An ALTER TABLE ... DROP COLUMN followed by an ALTER TABLE ... ADD COLUMN was required to change the expression. This was inconvenient when the column was referenced by other columns (DEFAULT, COMPUTED BY or AUTOMATIC AS expressions), routines, triggers, or constraints. Such references prevented the ALTER TABLE ... DROP COLUMN clause from succeeding.

With this release of Oracle Rdb, SQL will allow the COMPUTED BY or AUTOMATIC AS column expression to be revised by the ALTER TABLE ... ALTER COLUMN statement. To accommodate changes in data type and length of the data, a new row version is created for the table. If the column is an AUTOMATIC AS definition, then the previously stored data will be converted to the new data type upon retrieval.

The following example shows the changes now allowed by ALTER TABLE.

```
SQL> create table ttt (a integer, c computed by CURRENT_USER);
SQL> insert into ttt (a) values (10);
1 row inserted
SQL> select * from ttt;
           A   C
          10   SMITH
1 row selected
SQL>
SQL> show table (column) ttt
Information for table TTT
```

```
Columns for table TTT:
Column Name                     Data Type        Domain
-----------                     ---------        ------
A                               INTEGER
C                               CHAR(31)
         UNSPECIFIED 31 Characters,  31 Octets
 Computed:       by CURRENT_USER

SQL>
SQL> alter table ttt
cont>     alter c
cont>     computed by upper (substring (current_user from 1 for 1))
cont>           || lower (substring (current_user from 2));
SQL>
SQL> show table (column) ttt
Information for table TTT

Columns for table TTT:
Column Name                     Data Type        Domain
-----------                     ---------        ------
A                               INTEGER
C                               VARCHAR(31)
         UNSPECIFIED 31 Characters,  31 Octets
 Computed:       by upper (substring (current_user from 1 for 1))
                || lower (substring (current_user from 2))

SQL>
SQL> select * from ttt;
           A   C
          10   Smith
1 row selected
SQL>
```

This problem has been corrected in Oracle Rdb Release 7.1.4. Only the value expression can be altered; the type of computation cannot be changed. That is, a COMPUTED BY column cannot be changed to an AUTOMATIC AS column, an AUTOMATIC INSERT AS column cannot be changed to an AUTOMATIC UPDATE AS column, and so on.

4.1.5 Enhancements to CREATE and ALTER STORAGE MAP

With this release of Oracle Rdb, the CREATE STORAGE MAP and ALTER STORAGE MAP statements will create a SQL routine that matches the WITH LIMIT OF clause for the storage map.

This new routine will automatically be created in a special system module RDB$STORAGE_MAPS (use SHOW SYSTEM MODULES to view). The storage map name will be used to name the mapping routine (use SHOW SYSTEM FUNCTIONS to view).

These routines will be used by future releases of Rdb to validate the storage map definition. They are also available for customer applications that wish to process inserted data and determine the partition number of the new row.

The mapping routine returns the following values:

- Zero (0) if the storage map is defined as RANDOMLY ACROSS. This routine is just a descriptive place holder.

- Positive value representing the storage map number (the same value as stored in RDB$ORDINAL_POSITION column of the RDB$STORAGE_MAP_AREAS table). These values can be used with the PARTITION clause of the SET TRANSACTION ... RESERVING clause to reserve a specific partition prior to inserting the row.
- A −1 if the storage map has no OTHERWISE clause. This indicates that the row cannot be inserted as it doesn't match any of the WITH LIMIT OF clauses.

The following example shows the created routine from CREATE STORAGE MAP.

```
SQL> create table EMPLOYEES (
cont>     EMPLOYEE_ID      CHAR (5),
cont>     LAST_NAME        CHAR (14),
cont>     FIRST_NAME       CHAR (10),
cont>     MIDDLE_INITIAL   CHAR (1),
cont>     ADDRESS_DATA_1   CHAR (25),
cont>     ADDRESS_DATA_2   CHAR (25),
cont>     CITY             CHAR (20),
cont>     STATE            CHAR (2),
cont>     POSTAL_CODE      CHAR (5),
cont>     SEX              CHAR (1),
cont>     BIRTHDAY         DATE VMS,
cont>     STATUS_CODE      CHAR (1));
SQL>
SQL>     create storage map EMPLOYEES_MAP
cont>         for EMPLOYEES
cont>         comment is
cont>           ' employees partitioned by "00200" "00400"'
cont>         store
cont>             using (EMPLOYEE_ID)
cont>                 in EMPIDS_LOW
cont>                     with limit of ('00200')
cont>                 in EMPIDS_MID
cont>                     with limit of ('00400')
cont>                 otherwise in EMPIDS_OVER;
SQL>
SQL> commit work;
SQL>
SQL> show system modules;
Modules in database with filename MF_PERSONNEL
     RDB$STORAGE_MAPS
SQL>
SQL> show system functions;
Functions in database with filename MF_PERSONNEL
     EMPLOYEES_MAP
SQL>
SQL> show system function EMPLOYEES_MAP;
Information for function EMPLOYEES_MAP

 Function ID is: −2
 Source:
return
    case
        when (:EMPLOYEE_ID <= '00200') then 1
        when (:EMPLOYEE_ID <= '00400') then 2
        else 3
    end case;
 Comment:       Return value for select partition − range 1 .. 3
 Module name is: RDB$STORAGE_MAPS
 Module ID is: −1
 Number of parameters is: 1

Parameter Name                  Data Type        Domain or Type
--------------                  ---------        --------------
                                INTEGER
        Function result datatype
        Return value is passed by value

EMPLOYEE_ID                     CHAR(5)
        Parameter position is 1
        Parameter is IN (read)
        Parameter is passed by reference
```

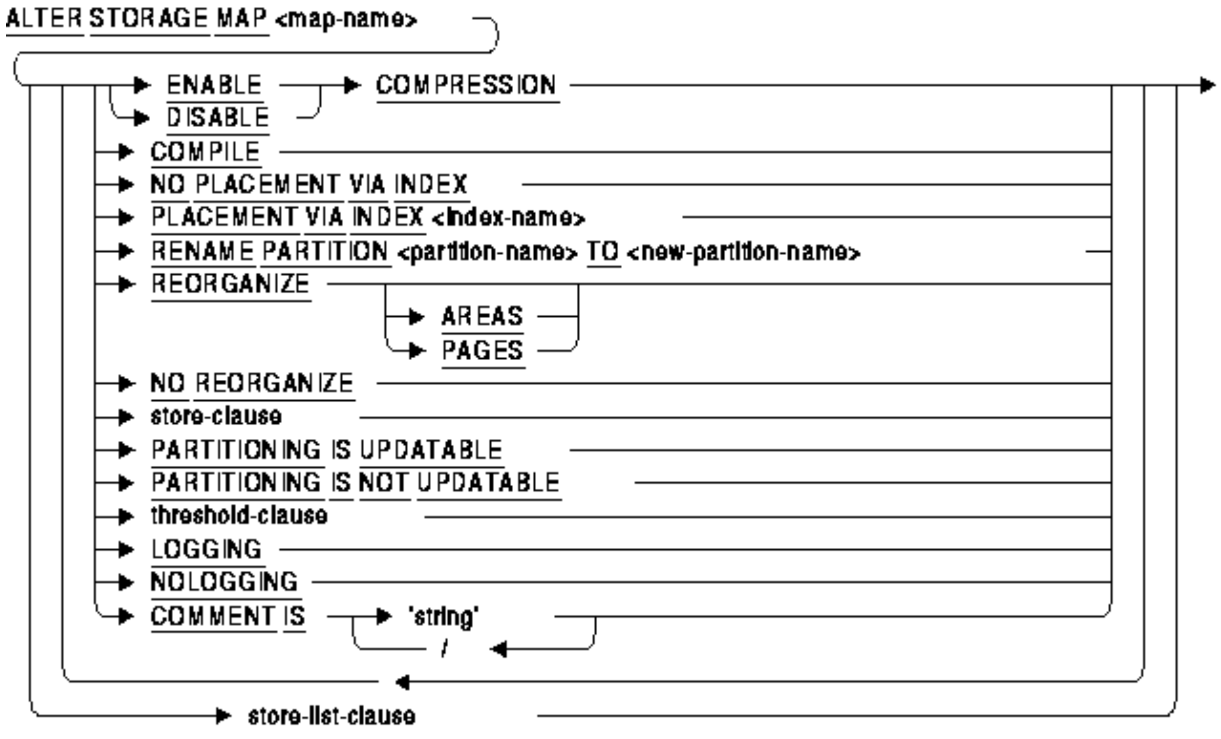
The ALTER STORAGE MAP command will remove any old mapping routine and redefine it when either the STORE clause is used or if the new COMPILE option is used.

```
SQL> alter storage map EMPLOYEES_MAP
cont>     store
cont>         using (EMPLOYEE_ID)
cont>             in EMPIDS_LOW
cont>                 with limit of ('00200')
cont>             in EMPIDS_MID
cont>                 with limit of ('00400')
cont>             in EMPIDS_OVER
cont>                 with limit of ('00800');
SQL>
SQL> show system function (source) EMPLOYEES_MAP;
Information for function EMPLOYEES_MAP

 Source:
return
    case
        when (:EMPLOYEE_ID <= '00200') then 1
        when (:EMPLOYEE_ID <= '00400') then 2
        when (:EMPLOYEE_ID <= '00800') then 3
        else −1
    end case;
```

The ALTER STORAGE MAP ... COMPILE statement can be used after installing this release of Rdb to create these mapping routines for the storage map.

Format


ALTER STORAGE MAP <map-name>
ENABLE
DISABLE
COMPRESSION
COMPILE
NO PLACEMENT VIA INDEX
PLACEMENT VIA INDEX <index-name>
RENAME PARTITION <partition-name> TO <new-partition-name>
REORGANIZE
AREAS
PAGES
NO REORGANIZE
store-clause
PARTITIONING IS UPDATABLE
PARTITIONING IS NOT UPDATABLE
threshold-clause
LOGGING
NOLOGGING
COMMENT IS
'string'
/
store-list-clause

Note

If a routine already exists with the same name as the storage map, the mapping routine will not be created.

If the storage map includes a STORE COLUMNS clause, that is a vertically partitioned map, then several routines will be created and uniquely named by adding the vertical partition number as a suffix.

Chapter 5
Enhancements Provided in Oracle Rdb Release 7.1.3

5.1 Enhancements Provided in Oracle Rdb Release 7.1.3

5.1.1 Dynamic Optimizer FAST FIRST Shortcut Termination

When the optimizer chose to use a dynamic retrieval strategy and the FAST FIRST retrieval strategy was used, query execution could involve redundant index scans and unnecessary I/O. The FAST FIRST tactic will now execute a type of shortcut termination to avoid unnecessary I/O.

When the retrieval strategy uses the FAST FIRST dynamic tactic, query execution starts by scanning the first background index. As dbkeys are read from the index, the rows are retrieved and if the row satisfies all the conditions on the query, the row is delivered. This continues until 1024 rows have been delivered. If 1024 rows are delivered, the FAST FIRST tactic is abandoned. The remaining index scans are executed and a complete dbkey list is constructed which is used to fetch the rows during the final phase of query execution.

During the final phase of execution, the complete dbkey list is first sorted and then each dbkey is processed by first checking to see if the FAST FIRST tactic may have already delivered that row. If the row has not already been delivered it is fetched, and if the row satisfies all the conditions on the query, the row is delivered.

This optimization is used in the case where the FAST FIRST tactic has not been abandoned. If a complete index scan has been performed, a type of shortcut termination is used to avoid unecessary I/O's.

The following example shows a query using the old behavior. Notice that the condition on birthday can never be true so the FAST FIRST tactic is going to fetch rows but never deliver any as no rows completely satisfy the query.

```
SQL> set flags 'strategy,detail,execution'
SQL> select * from employees where last_name>'L' and employee_id>'00200'
cont> and birthday<'01−Jan−1900';
~S#0006
Tables:
  0 = EMPLOYEES
Leaf#01 FFirst 0:EMPLOYEES Card=100
  Bool: (0.LAST_NAME > 'L') AND (0.EMPLOYEE_ID > '00200') AND (0.BIRTHDAY <
        '1−JAN−1900')
  BgrNdx1 EMP_EMPLOYEE_ID [1:0] Fan=17
    Keys: 0.EMPLOYEE_ID > '00200'
  BgrNdx2 EMP_LAST_NAME [1:0] Fan=12
    Keys: 0.LAST_NAME > 'L'
~Estim  EMP_EMPLOYEE_ID Sorted: Split lev=2, Seps=1 Est=17
~Estim  EMP_LAST_NAME Sorted: Split lev=2, Seps=3 Est=46
~E#0006.01(1) Estim   Index/Estimate 1/17 2/46
~E#0006.01(1) BgrNdx1 EofData  DBKeys=63  Fetches=0+0  RecsOut=0 #Bufs=24
~E#0006.01(1) BgrNdx2 EofData  DBKeys=30  Fetches=1+2  RecsOut=0 #Bufs=14
~E#0006.01(1) FgrNdx  FFirst   DBKeys=0  Fetches=0+10  RecsOut=0`ABA
~E#0006.01(1) Fin     Buf      DBKeys=30  Fetches=0+10  RecsOut=0
0 rows selected
```

The execution trace shows that the first background index (BgrNdx1) is scanned first. This means that the EMP_EMPLOYEE_ID index is scanned to find the dbkeys for all rows where EMPLOYEE_ID > '00200'. As each of the 63 dbkeys are read from the index, the FAST FIRST tactic means that the rows would be fetched and all the conditions for the query checked. In this case, there are no rows that satisfy the condition on birthday so no rows are delivered.

The second background index is then scanned to find the dbkeys for rows that have LAST_NAME > 'L'. The dbkeys from this index scan are only retained if they were also returned from the first index scan. At this point there are no more indexes to be scaned, so FAST FIRST (FgrNdx) is abandoned. The final (Fin) phase then uses the final dbkey list to fetch all the rows, test them, and if necessary deliver them.

During the first index scan, all rows where EMPLOYEE_ID > '00200' were fetched, tested and if possible delivered. So there is no possibility that any further rows need to be delivered. Any additional work performed to scan a second index or to fetch and filter records during the final (Fin) phase is redundant.

The dynamic optimizer has been enhanced to detect this case and avoid the unnecessary work of additional index scans and to avoid the final phase where FAST FIRST tactic is still running.

The following example shows the same query using the enhanced behavior.

```
SQL> set flags 'strat,detail,exec'
SQL> select * from employees where last_name>'L' and employee_id>'00200'
cont> and birthday<'01-Jan-1900';
~S#0003
Tables:
  0 = EMPLOYEES
Leaf#01 FFirst 0:EMPLOYEES Card=100
  Bool: (0.LAST_NAME > 'L') AND (0.EMPLOYEE_ID > '00200') AND (0.BIRTHDAY <
        '1-JAN-1900')
  BgrNdx1 EMP_EMPLOYEE_ID [1:0] Fan=17
    Keys: 0.EMPLOYEE_ID > '00200'
  BgrNdx2 EMP_LAST_NAME [1:0] Fan=12
    Keys: 0.LAST_NAME > 'L'
~Estim  EMP_EMPLOYEE_ID Sorted: Split lev=2, Seps=1 Est=17
~Estim  EMP_LAST_NAME Sorted: Split lev=2, Seps=3 Est=46
~E#0003.01(1) Estim   Index/Estimate 1/17 2/46
~E#0003.01(1) BgrNdx1 EofData  DBKeys=63  Fetches=0+0  RecsOut=0 #Bufs=24
~E#0003.01(1) FgrNdx  FFirst   DBKeys=0  Fetches=0+23  RecsOut=0`ABA
~E#0003.01(1) Fin     Buf_Ini  DBKeys=0  Fetches=0+0  RecsOut=0`ABA
0 rows selected
```

Notice that because FAST FIRST was still running when the first index scan completed, no further indexes were scanned. In addition, the final phase is abandoned because no further work is productive.

This enhancement should significantly reduce I/O and CPU costs where the FAST FIRST tactic is used and less than 1024 rows are delivered.

5.1.2 RDMS$DEBUG_FLAGS_OUTPUT Now Substitutes the Process ID in Output Filename

Enhancement 3633426

With this release of Oracle Rdb, the RDMS$DEBUG_FLAGS_OUTPUT logical name now allows substitution of the process id (PID) within the filename of the opened log file. When this logical name is translated, the result string is now processed and the first occurrence of the string _PID is replaced by the current OpenVMS process id. This string can appear in the directory, filename and file type portions of the file specification. The file specification can be in upper or lower case.

This change allows a system or group wide definition of the RDMS$DEBUG_FLAGS_OUTPUT logical name but also have each version of the log file identified by the executing user.

The following example shows the definition of the logical name.

```
$ Define RDMS$DEBUG_FLAGS_OUTPUT mf_personnel_pid.log
```

In the resulting log filename, the _PID is replaced by "_" and the process id in hexidecimal notation.

```
MF_PERSONNEL_220456C4.LOG;1
```

5.1.3 Peephole Optimization for Hidden Key Retrieval

In Oracle Rdb today, there are some operations which when executed on an indexed column, effectively hide that column from use by the optimizer and thus cause the optimizer to use a full scan instead of index lookup retrieval.

This problem is found in the OCI applications where the performance degrades significantly when most of the queries apply these type of operations that do not transform the data type of the underlying result data. It is most noticeable on data dictionary view queries because OCI requires space trimmed VARCHAR (31) result type. Queries on these views are forced to do full index scans, which on a database with many tables and views can slow down the application startup.

For example, the following simple query exhibits the desired strategy of index lookup:

```
SQL> select last_name from employees where last_name = 'Smith';
Tables:
  0 = EMPLOYEES
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [1:1]
    Keys: 0.LAST_NAME = 'Smith'
LAST_NAME
Smith
Smith
2 rows selected
```

When a CAST function is applied to the base indexed column, the optimizer switches to use a full scan [0:0] instead of index lookup [1:1] as in the above example:

```
SQL> select last_name from employees
where cast (last_name as varchar (31)) = 'Smith;
Tables:
  0 = EMPLOYEES
  Conjunct: CAST (0.LAST_NAME AS VARCHAR (31)) = 'Smith'
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [0:0]
LAST_NAME
Smith
Smith
2 rows selected
```

Even though the CAST function changes the original data type of the column "last_name" from CHAR (14) to VARCHAR (31), the underlying result data remains as a string of CHAR data type, filled with trailing spaces that is done by the equal operator before the comparison. Due to the fact that the underlying result data remains the same as the base indexed column, the optimizer should apply index lookup retrieval [1:1] for index search.

Another function that does not transform the underlying result data is TRIM (TRAILING), as seen in the following example:

```
SQL> select last_name from employees
where trim (trailing from last_name) = 'Smith';
Tables:
  0 = EMPLOYEES
Conjunct: TRIM (TRAILING ' ' FROM 0.LAST_NAME) = 'Smith'
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [0:0]
LAST_NAME
 Smith
 Smith
2 rows selected
```

Although the function TRIM (TRAILING) seems to transform the result data by trimming the trailing spaces, the equal operator fills the string back with the trailing spaces before comparison and thus the resultant data remains the same as the original data type CHAR (14) of the base indexed column.

Beside the equal operator, the following is a complete list of operators that blank fill the string with trailing spaces before comparison:

- equals (=)
- greater than (>)
- less than (<)
- greater or equal (>=)
- less or equal (<=)
- IS NULL
- IS NOT NULL

5.1.3.1 Feature Overview

A new feature of peephole optimization is implemented to peek into the predicate to see if these operators are present (may be deeply nested) and to use the hidden base column, if detected, as the index retrieval.

Our current approach is implemented inside the optimizer by finding the hidden base column and, if found, creating the index retrieval block (CRTV) accordingly for the predicate involving the hidden base column.

```
SQL> select last_name from employees
where cast (last_name as varchar (31)) = 'Smith;
Tables:
  0 = EMPLOYEES
Conjunct: TRIM (TRAILING ' ' FROM 0.LAST_NAME) = 'Smith'
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [1:1]   Hidden Key
    Keys: 0.LAST_NAME = 'Smith'
LAST_NAME
```

```
 Smith
 Smith
2 rows selected
```

Note that the notation "Hidden Key" is displayed to indicate that the key is retrieved by the index lookup scan performed on the index key (ikey) of the base column hidden under the CAST function.

It is important that this optimization should be performed specifically for CAST and TRIM (TRAILING) functions on the base indexed column of either CHAR or VARCHAR data type with the following operators that blank fill the string for comparison:

- equals (=)
- greater than (>)
- less than (<)
- greater or equal (>=)
- less or equal (<=)
- IS NULL
- IS NOT NULL

The expression "CAST (v1 as CHAR (n)) = v2" is equivalent to "v1 = v2" when the CHAR_LENGTH (v1) is less than the length of the CAST data type (n). Here the = operator blank fills v1 up to the length n before comparison.

The expression "TRIM (TRAILING from v1) = v2" is equivalent to "v1 = v2". Here the spaces are removed (trimmed) from v1 but the = operator just adds them back.

STARTING WITH and LIKE are the operators that are exceptions from the above blank filling characteristic, but they are also allowed for peephole optimization even though these operators do not blank fill the string before comparison.

The fixed leading characters of the pattern string are applied as the Boolean filter. For example, the following query exhibits the desired strategy with STARTING WITH:

```
SQL> select last_name from employees where
where last_name starting with 'Smith';
Tables:
  0 = EMPLOYEES
Conjunct: 0.LAST_NAME STARTING WITH 'Smith'
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [1:1]
    Keys: 0.LAST_NAME STARTING WITH 'Smith'
LAST_NAME
 Smith
 Smith
2 rows selected
```

When a CAST function is applied to the indexed column, the optimizer should not switch to use a full scan [0:0] as below:

```
SQL> select last_name from employees where
where where cast (last_name as varchar(31)) starting with 'Smith';
Tables:
  0 = EMPLOYEES
Conjunct: CAST (0.LAST_NAME AS VARCHAR(31)) STARTING WITH 'Smith'
```

```
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [0:0]
LAST_NAME
 Smith
 Smith
2 rows selected
```

With the new feature Peephole Optimization for Hidden Column Retrieval, the query will apply the index lookup retrieval, as follows:

```
SQL> select last_name from employees where
where where cast (last_name as varchar(31)) starting with 'Smith';
Tables:
  0 = EMPLOYEES
Conjunct: CAST (0.LAST_NAME AS VARCHAR(31)) STARTING WITH 'Smith'
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [1:1] Hidden Key
    Keys: 0.LAST_NAME STARTING WITH 'Smith'
LAST_NAME
 Smith
 Smith
2 rows selected
```

In this case, a low key value is constructed as '.Smith' from the pattern string 'Smith' and a high key value is constructed by adding one to the last byte of the low key value, e.g. '.Smiti'. These low and high key values are used in index lookup scan to find the range of index keys.

The LIKE operator will also be optimized in a similar way as follows:

```
SQL> select last_name from employees where
where where cast (last_name as varchar(31)) like 'Smith';
Tables:
  0 = EMPLOYEES
Conjunct: CAST (0.LAST_NAME AS VARCHAR(31)) LIKE 'Smith'
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [1:1] Hidden Key
    Keys: 0.LAST_NAME LIKE 'Smith'
0 rows selected
```

In this case, the low and high key values are constructed in the same way as in the case of STARTING WITH and the Boolean predicate of LIKE is applied as the filter after the index lookup scan [1:1].

Here is another interesting example with a multi−segmented index where all of the segment columns are used in index lookup scan as [3:3] as expected:

```
SQL>create index emp_last_first_mid on
    employees (last_name, first_name, middle_initial);

SQL>select last_name, first_name, middle_initial from employees where
    last_name = 'Smith' AND first_name = 'Roger' AND middle_initial = 'R';
Tables:
  0 = EMPLOYEES
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_FIRST_MID [3:3]
    Keys: (0.LAST_NAME = 'Smith') AND (0.FIRST_NAME = 'Roger') AND (
          0.MIDDLE_INITIAL = 'R')
 LAST_NAME        FIRST_NAME   MIDDLE_INITIAL
 Smith            Roger        R.
```

```
1 row selected
```

However, if we now apply TRIM (TRAILING) to all of the segment columns, the strategy will change to index full scan [0:0], as in the following example:

```
SQL>select last_name, first_name, middle_initial from employees where
        TRIM (TRAILING FROM last_name) = 'Smith' AND
        TRIM (TRAILING FROM first_name) = 'Roger' AND
        TRIM (TRAILING FROM middle_initial) = 'R';
Tables:
  0 = EMPLOYEES
Conjunct: (TRIM (TRAILING ' ' FROM 0.LAST_NAME) = 'Smith') AND (TRIM (TRAILING
          ' ' FROM 0.FIRST_NAME) = 'Roger') AND (TRIM (TRAILING ' ' FROM
          0.MIDDLE_INITIAL) = 'R')
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_FIRST_MID [0:0]
 LAST_NAME        FIRST_NAME   MIDDLE_INITIAL
 Smith            Roger        R.
1 row selected
```

With this new feature, the above query still does not change the original index lookup [3:3] applying the 'Hidden Key' retrieval strategy:

```
SQL>select last_name, first_name, middle_initial from employees where
        TRIM (TRAILING FROM last_name) = 'Smith' AND
        TRIM (TRAILING FROM first_name) = 'Roger' AND
        TRIM (TRAILING FROM middle_initial) = 'R' ;
Tables:
  0 = EMPLOYEES
Conjunct: (TRIM (TRAILING ' ' FROM 0.LAST_NAME) = 'Smith') AND (TRIM (TRAILING
          ' ' FROM 0.FIRST_NAME) = 'Roger') AND (TRIM (TRAILING ' ' FROM
          0.MIDDLE_INITIAL) = 'R')
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_FIRST_MID [3:3]   Hidden Key
    Keys: (0.LAST_NAME = 'Smith') AND (0.FIRST_NAME = 'Roger') AND (
          0.MIDDLE_INITIAL = 'R')
 LAST_NAME        FIRST_NAME   MIDDLE_INITIAL
 Smith            Roger        R.
1 row selected
```

This feature is always enabled by default. A new SQL flag called 'HIDDEN_KEY' is introduced to disable the feature in case the customer runs into some unexpected problem caused by this feature.

5.1.4 New PROTOTYPES Qualifier for SQL Module Language

With this release of Oracle Rdb, the SQL Module Language compiler supports a new /PROTOTYPES qualifier. This qualifier replaces the /C_PROTOTYPES qualifier from prior releases and supports prototypes (routine declarations) generation for C (C++), Pascal and BLISS.

- /PROTOTYPES[=prototypesfile]
  The PROTOTYPES qualifier uses the LANGUAGE clause from the module to generate routine declarations for the following languages: C (C++), Pascal and BLISS. The qualifier is ignored for all other language values.
  The prototypes file specification defaults to the same device, directory, and file name as the module language source. The file types default to .h for C, .PAS for Pascal and .REQ for BLISS.

The default is /NOPROTOTYPES.
- /C_PROTOTYPES qualifier is now deprecated and is now a synonym for /PROTOTYPES
- Language BLISS has been added. It is similar to LANGUAGE GENERAL. The /PROTOTYPES qualifier will generate EXTERNAL ROUTINE declarations for each SQL module language procedure.
- Language PASCAL support has been added. The generated external procedure declarations are suitable for inclusion either in a Pascal program or module. Structured types (RECORD ... END RECORD), SQLDA and SQLCA used by the SQL module language procedures are declared as UNSAFE arrays of bytes to simplify passing structures via these external definitions. However, care must be taken as this form of declaration disables the Pascal strong typing checks.
- The output for LANGUAGE C has been enhanced to include pre−processor directives to conditionally include C++ "extern C" syntax and also allow multiple #include references.

5.1.5 RMU /UNLOAD /AFTER_JOURNAL Performance Enhancement

The RMU /UNLOAD /AFTER_JOURNAL command buffers all content for a transaction as the after−image journal is read. When a "commit" record is found, all records for the transaction are sorted to allow duplicate removal such that the final record content is returned. After the sort phase, only those records selected for extraction are returned to the output stream.

In some cases of processing transactions that perform modifications to record types (including index nodes) not selected for extraction, the overhead of buffering, sorting and removing the records not selected can become excessive.

The impact of this situation has been reduced. The LogMiner is now able to more effectively filter records that cannot be extracted as they are being read from the after−image journal. This filtering improves performance by avoiding buffering and sorting of those records that are known to not match the selection criteria.

5.1.6 New GET DIAGNOSTICS Keywords

The following new keywords have been added to GET DIAGNOSTICS:

- LIMIT_CPU_TIME
  Returns an INTEGER value for the session's execution CPU time limit in seconds. If zero (0) is returned then that is equivalent to no CPU time limit. This value is established by either the logical name RDMS$BIND_QG_EXEC_CPU_TIMEOUT or the SET QUERY EXECUTION LIMIT CPU TIME statement.
- LIMIT_ROWS_FETCHED
  Returns a BIGINT value for the session's row limit. If zero (0) is returned then that is equivalent to no row limit. This value is established by the logical name RDMS$BIND_QG_REC_LIMIT.
- LIMIT_ELAPSED_TIME
  Returns an INTEGER value for the session's execution elapsed time limit in seconds. If zero (0) is returned then that is equivalent to no elapsed time limit. This value is established by either the logical name RDMS$BIND_QG_EXEC_ELAPSED_TIMEOUT or the SET QUERY EXECUTION LIMIT ELAPSED TIME statement.

The following example shows usage of these keywords in a compound statement.

```
SQL> set flags 'trace';
SQL> set query execution limit elapsed time 10 minutes;
SQL> begin
cont> declare :row_limit integer;
cont> declare :elapsed_limit integer;
cont> declare :cpu_limit integer;
cont> get diagnostics
cont>     :cpu_limit = LIMIT_CPU_TIME,
cont>     :row_limit = LIMIT_ROWS_FETCHED,
cont>     :elapsed_limit = LIMIT_ELAPSED_TIME;
cont> trace 'LIMIT_ROWS_FETCHED: ', :row_limit;
cont> trace 'LIMIT_CPU_TIME:     ', :cpu_limit;
cont> trace 'LIMIT_ELAPSED_TIME: ', :elapsed_limit;
cont> end;
~Xt: LIMIT_ROWS_FETCHED: 0
~Xt: LIMIT_CPU_TIME:     0
~Xt: LIMIT_ELAPSED_TIME: 600
SQL>
```

5.1.7 RMU /BACKUP New /ALLOCATION_QUANTITY Qualifier

An "/ALLOCATION_QUANTITY=number−blocks" qualifier has been added to the RMU /BACKUP command. This qualifier specifies the size, in blocks, which the backup file is initially allocated. The minimum value for the number−blocks parameter is 1; the maximum value allowed is 2147483647. If you do not specify the ALLOCATION_QUANTITY qualifier, the EXTEND_QUANTITY value effectively controls the file's initial allocation.

The ALLOCATION_QUANTITY qualifier cannot be used with backup operations to tape. Dynamically executes a previously prepared statement.

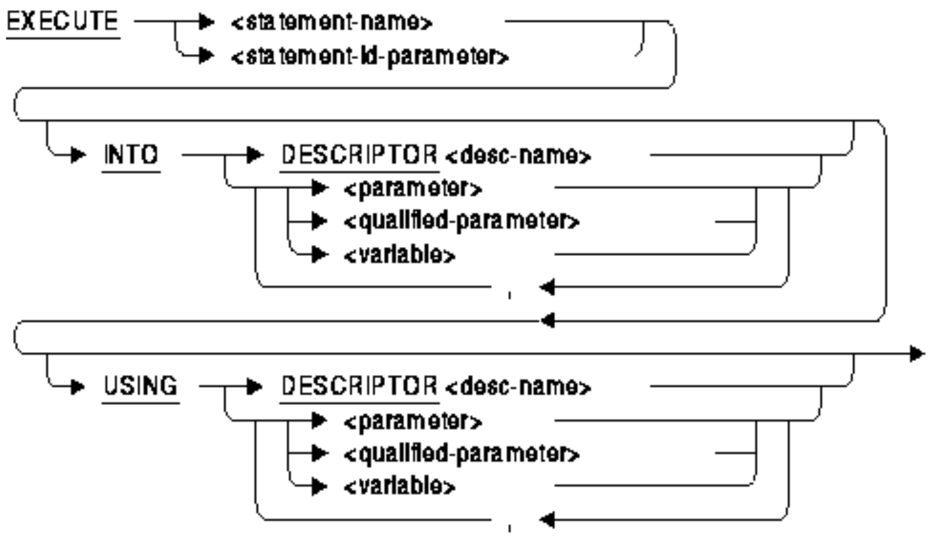
5.1.8 New EXECUTE Syntax

The EXECUTE statement is a dynamic SQL statement. Dynamic SQL lets programs accept or generate SQL statements at run time, in contrast to SQL statements that are part of the source code for precompiled programs or SQL module language procedures. Unlike precompiled SQL or SQL module language statements, such dynamically executed SQL statements are not necessarily part of a program's source code, but can be generated while the program is running. Dynamic SQL is useful when you cannot predict the type of SQL statement your program will need to process.

If a program needs to dynamically execute a statement more than once, the statement should be prepared first with the PREPARE statement and executed each time with the EXECUTE statement. SQL does not parse and compile prepared statements every time it dynamically executes them with the EXECUTE statement.

You can use the EXECUTE statement:

- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module



ARGUMENTS

INTO DESCRIPTOR descriptor−name

Specifies an SQLDA descriptor that contains addresses and data types that specify output parameters or variables.

The descriptor must be a structure declared in the host language program as an SQLDA. If the program is precompiled and uses the embedded SQL statement INCLUDE SQLDA, the name of the structure is simply SQLDA. Programs can use multiple SQLDAs, but must explicitly declare them with names other than SQLDA.

Programs can always use the INTO DESCRIPTOR clause of the EXECUTE statement whether or not the statement string contains output parameter markers, as long as the value of the SQLD field in the SQLDA corresponds to the number of output parameter markers. SQL updates the SQLD field with the correct number of output parameter markers when it processes the DESCRIBE statement for the statement string.

INTO parameter

INTO qualified−parameter

INTO variable

Specifies output parameters or variables whose values are returned by a successful EXECUTE statement.

When you specify a list of parameters or variables, the number of parameters in the list must be the same as the number of output parameter markers in the statement string of the prepared statement. If SQL determines that a statement string had no output parameter markers, the INTO clause is not allowed.

statement−name

statement−id−parameter

Specifies the name of a prepared statement. You can supply either a parameter or a compile−time statement name. Specifying a parameter lets SQL supply identifiers to programs at run time. Use an integer parameter to contain the statement identifier returned by SQL or a character string parameter to contain the name of the statement that you pass to SQL.

If the PREPARE statement for the dynamically executed statement specifies a parameter, use that same parameter in the EXECUTE statement instead of an explicit statement name.

USING DESCRIPTOR descriptor−name

Specifies an SQLDA descriptor that contains addresses and data types of input parameters or variables.

The descriptor must be a structure declared in the host language program as an SQLDA. If the program is precompiled and uses the embedded SQL statement INCLUDE SQLDA, the name of the structure is simply SQLDA. Programs can use multiple SQLDAs, but must explicitly declare them with names other than SQLDA.

Programs can always use the USING DESCRIPTOR clause of the EXECUTE statement whether or not the statement string contains input parameter markers, as long as the value of the SQLD field in the SQLDA corresponds to the number of input parameter markers. SQL updates the SQLD field with the correct number of input parameter markers when it processes the DESCRIBE statement for the statement string.

USING parameter

USING qualified−parameter

USING variable

Specifies input parameters or variables whose values SQL uses to replace parameter markers in the prepared statement string.

When you specify a list of parameters or variables, the number of parameters in the list must be the same as the number of input parameter markers in the statement string of the prepared statement. If SQL determines that a statement string had no input parameter markers, the USING clause is not allowed.

Usage Notes

- You must use at least one USING or one INTO clause in an EXECUTE statement. If the statement has no parameters then use the EXECUTE IMMEDIATE statement instead.

- You may mix parameters with DESCRIPTOR structures within the EXECUTE statement. That is, you may use INTO DESCRIPTOR to hold the results of the dynamic statement, but use USING with a list of parameters to provide the input values.
- When you issue the EXECUTE statement for a previously prepared statement, you might want to obtain information beyond the success or failure code returned in the SQLCODE status parameter. For example, you might want to know how many rows were affected by the execution of an INSERT, DELETE, UPDATE, FETCH, or SELECT statement. SQL returns this information in the SQLERRD[2] field of the SQLCA.
  However, when you use an SQLCA parameter to prepare a statement, you must also use an SQLCA parameter when you execute that statement. For example, using SQL module language calls from C, your code might look like the following where the SQLCA parameter is passed to both procedures:

  ```
  static struct SQLCA  sqlca;
  /* ... */
  PREPARE_STMT(&sqlca, statement, &stmt_id);
  /* ... */
  EXECUTE_STMT(&sqlca, &stmt_id);
  ```

  For more information about the SQLCA, including the SQLERRD[2] field, see Oracle Rdb SQL Reference Manual.

Examples

Example 1: Executing an INSERT statement with parameter markers

These fragments from the online sample C program sql_dynamic illustrate using an EXECUTE statement in an SQL module procedure to execute a dynamically generated SQL statement.

The program accepts input of any valid SQL statement from the terminal and calls the subunit shown in the following program excerpt:

```
   .
   .
   .
/*
**----------------------------------------------------------------------------
**  Begin Main routine
**----------------------------------------------------------------------------
*/


  int   sql_dynamic (psql_stmt, input_sqlda, output_sqlda, stmt_id, is_select)
  char  *psql_stmt;
  sqlda *input_sqlda;
  sqlda *output_sqlda;
  long  *stmt_id;
  int   *is_select;

{
    sqlda sqlda_in, sqlda_out;    /* Declare the SQLDA structures. */
    int rowcount, status;
    int param;


/* Declare arrays for storage of original data types and allocate memory. */
```

```
mem_ptr output_save;
mem_ptr input_save;

/* * If a NULL SQLDA is passed, then a new statement is being prepared. */

if ((*input_sqlda == NULL) && (*output_sqlda == NULL))
     {
     new_statement = TRUE;

     /*
     * Allocate separate SQLDAs for input parameter markers (SQLDA_IN)
     * and output list items (SQLDA_OUT).  Assign the value of the constant
     * MAXPARMS to the SQLN field of both SQLDA structures.  SQLN specifies
     * to SQL the maximum size of the SQLDA.
     */

    if ((sqlda_in = (sqlda) calloc (1, sizeof (sqlda_rec))) == 0)
        {
        printf ("\n\n*** Error allocating memory for sqlda_in: Abort");
        return (-1);
        }
    else    /* set # of possible parameters */
        sqlda_in->sqln = MAXPARAMS;

    if ((sqlda_out = (sqlda) calloc (1, sizeof (sqlda_rec))) == 0)
        {
        printf ("\n\n*** Error allocating memory for sqlda_out: Abort");
        return (-1);
        }

        }
    else
        /* Set # of possible select list items. */
        sqlda_out->sqln = MAXPARAMS;
    /* copy name SQLDA2 to identify the SQLDA */

    strncpy(&sqlda_in->sqldaid[0],"SQLDA2  ",8);
    strncpy(&sqlda_out->sqldaid[0],"SQLDA2  ",8);

    /*
    * Call an SQL module language procedure, prepare_stmt and
    * describe_stmt that contains a PREPARE and DESCRIBE...OUTPUT
    * statement to prepare the dynamic statement and write information
    * about any select list items in it to SQLDA_OUT.
    */

    *stmt_id = 0;  /* If <> 0 the BADPREPARE error results in the PREPARE.*/

    PREPARE_STMT (&SQLCA, stmt_id, psql_stmt);
    if (SQLCA.SQLCODE  != sql_success)
        {
        printf ("\n\nDSQL-E-PREPARE, Error %d encountered in PREPARE",
                    SQLCA.SQLCODE);
        display_error_message();
        return (-1);
        }

    DESCRIBE_SELECT (&SQLCA, stmt_id, sqlda_out);
    if (SQLCA.SQLCODE  != sql_success)
        {
```

```
        printf ("\n\nDSQL−E−PREPARE, Error %d encountered in PREPARE",
                    SQLCA.SQLCODE);
        display_error_message();
        return (−1);
        }
    /*
    * Call an SQL module language procedure, describe_parm, that contains a
    * DESCRIBE...INPUT statement to write information about any parameter
    * markers in the dynamic statement to sqlda_in.
    */

    DESCRIBE_PARM (&SQLCA, stmt_id, sqlda_in);
    if (SQLCA.SQLCODE  != sql_success)
        {
        printf ("\n\n*** Error %d returned from describe_parm: Abort",
                 SQLCA.SQLCODE);
        display_error_message();
        return (−1);
        }
    /* Save the value of the SQLCA.SQLERRD[1] field so that program can
     *  determine if the statement is a SELECT statement or not.
     *  If the value is 1, the statement is a SELECT statement.*/

       *is_select = SQLCA.SQLERRD[1];
        .
        .
        .

/*
* Check to see if the prepared dynamic statement contains any parameter
* markers by looking at the SQLD field of sqlda_in.  SQLD contains the
* number of parameter markers in the prepared statement. If SQLD is
* positive, the prepared statement contains parameter markers.  The program
* executes a local procedure, get_in_params, that prompts the user for
* values, allocates storage for those values, and updates the SQLDATA field
* of sqlda_in:
*/

if (sqlda_in−>sqld > 0)
    if ((status = get_in_params(sqlda_in,input_save)) != 0)
        {
        printf ("\nError returned from GET_IN_PARAMS. Abort");
        return (−1);
        }

/* Check to see if the prepared dynamic statement is a SELECT by looking
* at the value in is_select, which stores the value of the
* SQLCA.SQLERRD[1] field.  If that value is equal to 1, the prepared
* statement is a SELECT statement.  The program allocates storage for
* rows for SQL module language procedures to open and fetch from a cursor,
* and displays the rows on the terminal:
*/

if (*is_select)
    {
    if (new_statement == TRUE)      /* Allocate buffers for output. */
        {
        /* assign a unique name for the cursor */
        sprintf(cursor_name,"%2d",++cursor_counter);

        if ((status = allocate_buffers(sqlda_out)) != 0)
     .
     .
     .

/*
* If the SQLCA.SQLERRD[1] field is not 1, then the prepared statement is not a
* SELECT statement and only needs to be executed.  Call an SQL module language
* procedure to execute the statement, using information about parameter
* markers stored in sqlda_in by the local procedure get_in_params:
*/
        {
        EXECUTE_STMT (&SQLCA, stmt_id, sqlda_in);
        if (SQLCA.SQLCODE != sql_success)
   .
   .
   .
```

The SQL module language procedures called by the preceding fragment:

```
   .
   .
   .
-------------------------------------------------------------------------------
-- Procedure Section
-------------------------------------------------------------------------------

-- This procedure prepares a statement for dynamic execution from the string
-- passed to it.  It also writes information about the number and data type of
-- any select list items in the statement to an SQLDA2 (specifically,
-- the sqlda_out SQLDA2 passed to the procedure by the calling program).
--

PROCEDURE PREPARE_STMT
    SQLCA
    :DYN_STMT_ID      INTEGER
    :STMT            CHAR(1024);

    PREPARE :DYN_STMT_ID FROM :STMT;

-- This procedure writes information to an SQLDA (specifically,
-- the sqlda_in SQLDA passed to the procedure by the calling program)
-- about the number and data type of any parameter markers in the
-- prepared dynamic statement.  Note that SELECT statements may also
-- have parameter markers.

PROCEDURE DESCRIBE_SELECT
    SQLCA
    :DYN_STMT_ID    INTEGER
    SQLDA;

    DESCRIBE :DYN_STMT_ID OUTPUT INTO SQLDA;

PROCEDURE DESCRIBE_PARM
    SQLCA
    :DYN_STMT_ID INTEGER
    SQLDA;

    DESCRIBE :DYN_STMT_ID INPUT INTO SQLDA;
```

```
-- This procedure dynamically executes a non−SELECT statement.
-- SELECT statements are processed by DECLARE CURSOR, OPEN CURSOR,
-- and FETCH statements.
--
-- The EXECUTE statement specifies an SQLDA2 (specifically,
-- the sqlda_in SQLDA2 passed to the procedure by the calling program)
-- as the source of addresses for any parameter markers in the dynamic
-- statement.
--
-- The EXECUTE statement with the USING DESCRIPTOR clause
-- also handles statement strings that contain no parameter markers.
-- If a statement string contains no parameter markers, SQL sets
-- the SQLD field of the SQLDA2 to zero.

PROCEDURE EXECUTE_STMT
    SQLCA
    :DYN_STMT_ID INTEGER
    SQLDA;

    EXECUTE :DYN_STMT_ID USING DESCRIPTOR SQLDA;
   .
   .
   .
```

5.1.9 Bitmap Scan Performance Enhancements

Several enhancements have been made to the dynamic optimizer when the bitmap scan feature is enabled. These enhancements can improve I/O and CPU times in various situations when the bitmap scan feature is enabled.

5.1.9.1 Bitmap Scan for OR Index Retrieval

Enhancement 3321352

The ability to use bitmap scan functionality has been enhanced to include queries that previously used a static "or" tactic.

During query compilation where the query contains conditions combined using the *OR* keyword, the optimizer can decide that the conditions in the query can be satisfied by scanning multiple indexes and combining the results. This is termed "or index retrieval" or "static or".

The following example shows a query that uses a static OR tactic.

```
SQL> create index sexi on employees (sex) type is sorted ranked;
SQL> create index addi on employees (address_data_1) type is sorted ranked;
SQL> commit;
SQL> set flags 'stratgey,detail'
SQL> select count(*) from employees
cont>     where address_data_1 starting with '1' or sex='M';
Tables:
  0 = EMPLOYEES
Aggregate: 0:COUNT (*)
OR index retrieval
  Conjunct: 0.ADDRESS_DATA_1 STARTING WITH '1'
  Get     Retrieval by index of relation 0:EMPLOYEES
```

```
    Index name  ADDI [1:1]
      Keys: 0.ADDRESS_DATA_1 STARTING WITH '1'
  Conjunct: NOT (0.ADDRESS_DATA_1 STARTING WITH '1')
  Get     Retrieval by index of relation 0:EMPLOYEES
    Index name  SEXI [1:1]
      Keys: 0.SEX = 'M'

          81
1 row selected
```

During query execution of a static or, the first index is scanned and these rows are delivered. When the second index is scanned, all rows are fetched, but only those rows not already delivered by the first scan are delivered. Rdb places a test on the second (and any subsequent) index scan to exclude rows that would have been delivered by previous indices. In this case, the test on the second index is (NOT (0.ADDRESS_DATA_1 STARTING WITH '1')). In this way, any rows that satisfy both of the OR conditions will be fetched twice. In addition, the rows are read in the order that they appear in the index, which most often means that acess to the data is physically random.

By using a bitmap scan, an in−memory BBC bitmap is constructed for dbkeys from the first index scan. A second BBC bitmap is constructed for dbkeys from the second index scan. These two bitmaps are then combined using a logical OR operation. The resulting bitmap is used to retrieve the rows. This has the advantage of retrieving all the rows selected only once, and in addition, since bitmaps are logically sorted, rows are fetched in order of their physical location (dbkey).

In the following example, we can see that with bitmap scan enabled the strategy chosen is a dynamic tactic with multiple "or" indexes and that bitmapped scan is being used.

```
SQL> set flags 'bitmapped_scan'
SQL> select count(*) from employees
cont>     where address_data_1 starting with '1' or sex='M';
Tables:
  0 = EMPLOYEES
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:EMPLOYEES Card=100     Bitmapped scan
  Bool: (0.ADDRESS_DATA_1 STARTING WITH '1') OR (0.SEX = 'M')
  BgrNdx1 ADDI [1:1] Fan=9
    Keys: 0.ADDRESS_DATA_1 STARTING WITH '1'
      OrNdx1 SEXI [1:1] Fan=19
        Keys: 0.SEX = 'M'

          81
1 row selected
```

In the past, this type of query would have been executed using an alternate tactic. In most cases, this would have been a static "or" tactic. The use of bitmap scan had previously required an additional condition with an "and" on an index field. As with other bitmap scan strategies, the strategy depends on the presence of at least one index of *TYPE IS SORTED RANKED.*

In this example, the use of bitmap scan reduced the number of I/O's from 81 to 54. This represents an approximate 30% reduction in I/O for this simple query.

The use of bitmap scan is best explained in the Rdb Journal article titled "Guide to Database Performance and Tuning: Bitmapped Scan".

5.1.9.2 Processing of Unique Keys and Non−ranked Indices

When scanning an index that did not have BBC duplicates, either because the index was not of *TYPE IS SORTED RANKED*, or because the index was unique, the dynamic optimizer would add one dbkey at a time into an in−memory BBC. This proves to be very CPU intensive.

The following example shows the execution trace from a query where bitmap scan is enabled.

```
SQL> set flags 'strategy,execut,bitmapped_scan,detail(1)'
SQL> select a1,f0,f1 from t
cont> where a1<30 and a2<30 and a3=1
cont> and f0>5 and f0<11
cont> optimize for total time;
~S#0001
Tables:
  0 = T
Leaf#01 BgrOnly 0:T Card=3000    Bitmapped scan
  Bool: (0.A1 < 30) AND (0.A2 < 30) AND (0.A3 = 1) AND (0.F0 > 5)
         AND (0.F0 < 11)
  BgrNdx1 I3 [1:1] Fan=1
    Keys: 0.A3 = 1
  BgrNdx2 I1 [0:1] Fan=17
    Keys: 0.A1 < 30
  BgrNdx3 I2 [0:1] Fan=17
    Keys: 0.A2 < 30
~Estim  I3 Hashed: Nodes=0, Est=1 Disabled IO=0
~Estim  I1 Ranked: Nodes=1, Min=29, Est=29 Precise IO=2
~Estim  RLEAF Cardinality=  2.9380000E+03
~Estim  I2 Sorted: Split lev=1, Seps=29 Est=29 Precise
~E#0001.01(1) Estim   Index/Estimate 1_1 2/29 3/29
~E#0001.01(1) BgrNdx1 FillMap2  DBKeys=1 Fetches=1+0
~E#0001.01(1) BgrNdx1 FillMap2  DBKeys=1 Fetches=0+0
~E#0001.01(1) BgrNdx1 Or__Map2  DBKeys=2 Fetches=0+0
~E#0001.01(1) BgrNdx1 FillMap2  DBKeys=1 Fetches=0+0
~E#0001.01(1) BgrNdx1 Or__Map2  DBKeys=3 Fetches=0+0
.
.
.
~E#0001.01(1) BgrNdx1 FillMap2  DBKeys=1 Fetches=0+0
~E#0001.01(1) BgrNdx1 Or__Map2  DBKeys=19 Fetches=0+0
~E#0001.01(1) BgrNdx1 EofData  DBKeys=19  Fetches=0+0  RecsOut=0 #Bufs=0
~E#0001.01(1) BgrNdx2 FtchLim  DBKeys=0  Fetches=0+0  RecsOut=0
          A1            F0   F1
           6             6   test
           7             7   test
           8             8   test
           9             9   test
~E#0001.01(1) Fin     Bitmap   DBKeys=19  Fetches=0+0  RecsOut=5
          10            10   test
5 rows selected
```

Notice from the index estimation execution trace line that the first background index, I3, is of *TYPE IS HASHED*. Since a hashed index does not have BBC duplicates chains, each dbkey is added one at a time into an in−memeory BBC. The new in−memory BBC is then logically OR'ed with dbkeys previously fetched and the two old BBCs are deleted.

Rdb now detects the case where there is no BBC in the index being scanned and reads the dbkeys into an in−memory buffer of 1024 dbkeys. When the index scan completes or when the 1024 dbkey buffer fills up, the dbkeys are sorted and rolled into an in−memory BBC.

The next example shows the same query with the enhanced behavior. This example starts at the execution estimation summary line as all prior information is the same as the preceeding example.

```
~E#0001.01(1) Estim   Index/Estimate 1_1 2/29 3/29
~E#0001.01(1) BgrNdx1 EofData  DBKeys=19  Fetches=1+0  RecsOut=0 #Bufs=0
~E#0001.01(1) BgrNdx1 Bld_Map2  DBKeys=19 Fetches=0+0
~E#0001.01(1) BgrNdx2 FtchLim  DBKeys=0  Fetches=0+0  RecsOut=0
          A1            F0   F1
           6             6   test
           7             7   test
           8             8   test
           9             9   test
~E#0001.01(1) Fin     Bitmap   DBKeys=19  Fetches=0+0  RecsOut=5
          10            10   test
5 rows selected
```

The new behavior shows the first background index is scanned to completion. Because the index is hashed, the dbkeys are collected in the in−memory buffer. Only when the index scan completes are the dbkeys sorted and then rolled into an in−memory BBC. This is shown by the *BgrNdx1 Bld_Map2* execution trace line. This can significantly reduce the CPU time used during bitmap scans on indexes that do not have BBC duplicates chains. Tests have shown a 75% reduction in CPU time for some queries.

Where the index being scanned is of *TYPE IS SORTED RANKED* and the index contains a mixture of unique and duplicate keys, each duplicate will be handled by building an in−memory BBC for the dbkeys of the duplicate key and logically OR'ing that with previously fetched dbkeys, as before. However, where the dbkey is unique, it will be added to the in−memory dbkey buffer until the scan completes or the dbkey buffer fills up. At that time, the dbkey buffer will be rolled into an in−memory BBC and logically OR'ed with any existing in−memory BBC.

5.1.9.3 Enhanced Fast First Processing

In the fast first tactic (FFirst), Rdb attempts to deliver the first 1024 rows as quickly as possible. To do this, the first background index is scanned and as each dbkey is fetched from the index scan, it is passed to foreground (Fgr). Foreground will fetch the row, check that all conditions on the query are met, and if necessary deliver the row.

Foreground maintains a list of all delivered dbkeys in an in−memory 1024 dbkey buffer. If this buffer fills up, then foreground is abandoned (ABA) and execution reverts to a background only tactic (BgrOnly). The foreground dbkey list is retained to ensure that already delivered rows are not delivered a second time.

In the past, Rdb has always retained a background dbkey list. This could be an in−memory 1024 dbkey buffer or a temporary file on disk depending on how many dbkeys had been fetched from the index. With bitmap scan, this was maintained as an in−memory BBC bitmap.

Since dbkeys are processed one row at a time, these dbkeys must be inserted into the background dbkey list one at a time regardless of the index type and structure being scanned. If the background index scan completes and fast first is still running, we can be sure that all rows necessary have been delivered by foreground, so execution is abandoned. In this case, the background dbkey list was not used.

If 1024 rows are delivered by foreground then fast first is abandoned. In this case, the background dbkey list is completed by finishing the index scan. The final phase (FIN) uses the background dbkey list to fetch the rows that are not in the foreground dbkey list and, if necessary, delivers the rows. However, dbkeys in the background dbkey list at the time that fast first was abandoned have already been fetched and delivered by foreground. By retaining them in the background dbkey list, we potentially cause the final phase to do extra work to re−fetch and filter these rows.

When using the bitmap scan feature, Rdb no longer maintains a background dbkey BBC if fast first is running. In this way, a potentially large amount of CPU is saved on BBC operations and the number of dbkeys needing to be processed by FIN is reduced.

There is one exception to this. When using the bitmap scan feature, Rdb can use an OR index list for an index scan. In the example below, both of the indexes return the same dbkey for the Alvin Toliver row. In this case, background must maintain the background in−memory BBC to ensure that the same row is never delivered twice. So each new dbkey is inserted into the existing BBC as it is read. Before being inserted, the BBC is checked to ensure that this dbkey has not already been delivered. If it has already been delivered by a previous "or" index, it is ignored.

```
SQL> select * from employees
cont> where last_name='Toliver' or first_name='Alvin';
~S#0005
Tables:
  0 = EMPLOYEES
Leaf#01 FFirst 0:EMPLOYEES Card=100      Bitmapped scan
  Bool: (0.LAST_NAME = 'Toliver') OR (0.FIRST_NAME = 'Alvin')
  BgrNdx1 EMP_LAST_NAME [1:1] Fan=12
    Keys: 0.LAST_NAME = 'Toliver'
      OrNdx1 EMP_FIRST_NAME [1:1] Fan=14
        Keys: 0.FIRST_NAME = 'Alvin'
~Estim  EMP_LAST_NAME Sorted: Split lev=1, Seps=1 Est=1 Precise
~Estim  EMP_FIRST_NAME Ranked: Nodes=1, Min=2, Est=2 Precise IO=0
~E#0005.01(1) Estim   Index/Estimate 1/2
~E#0005.01(1) BgrNdx1 EofData  DBKeys=1  Fetches=0+0  RecsOut=1 #Bufs=0
 EMPLOYEE_ID   LAST_NAME        FIRST_NAME   MIDDLE_INITIAL
   ADDRESS_DATA_1              ADDRESS_DATA_2         CITY
      STATE   POSTAL_CODE   SEX    BIRTHDAY      STATUS_CODE
 00164         Toliver          Alvin        A
   146 Parnell Place                                  Chocorua
      NH      03817         M      28−Mar−1947   1

~E#0005.01(1) Or Ndx1 EofData  DBKeys=1  Fetches=0+0  RecsOut=2 #Bufs=0
~E#0005.01(1) FgrNdx  FFirst   DBKeys=2  Fetches=0+1  RecsOut=2`ABA
~E#0005.01(1) Fin     Bitmap   DBKeys=0  Fetches=0+0  RecsOut=2`ABA
 00405         Dement           Alvin        B
   101 Second St.                                     Sanbornton
      NH      03269         M       7−Aug−1931   1

2 rows selected
```

If the first background index is not an "or" index list, then background does not retain a dbkey list while fast first is running. If the first background index is an "or" index list then the BBC is maintained; however, it is discarded if foreground is abandoned. FIN will use foreground's 1024 dbkey list to ensure that no row is delivered twice.

In the following example, we see where fast first is abandoned (ABA). Both the indexes used are made unique by including the field F3 as a second field in the index. Note that 2000 rows are selected by this query.

```
SQL> set flags 'strat,detail,exec,bitmap'
SQL> select * from t1 where f1=1 and f2=1;
~S#0007
Tables:
  0 = T1
Leaf#01 FFirst 0:T1 Card=2000    Bitmapped scan
  Bool: (0.F1 = 1) AND (0.F2 = 1)
  BgrNdx1 I11 [1:1] Fan=14
    Keys: 0.F1 = 1
  BgrNdx2 I12 [1:1] Fan=14
    Keys: 0.F2 = 1
~Estim  I11 Ranked: Nodes=88, Min=713, Est=1696 IO=1
~Estim  RLEAF Cardinality=  2.0000000E+03
~Estim  I12 Ranked: Nodes=88, Min=713, Est=1696 IO=2
~E#0007.01(1) Estim   Index/Estimate 1/1696 2/1696
          F1            F2            F3
           1             1             1
           1             1             2
           1             1             3
           1             1             4
           1             1             5
.
.
.
           1             1          1022
           1             1          1023
~E#0007.01(1) FgrNdx  FFirst   DBKeys=1024  Fetches=0+15  RecsOut=1024`ABA
~E#0007.01(1) BgrNdx1 Bld_Map2  DBKeys=1 Fetches=1+8
~E#0007.01(1) BgrNdx1 EofData  DBKeys=975  Fetches=0+13  RecsOut=1024 #Bufs=0
~E#0007.01(1) BgrNdx1 Bld_Map2  DBKeys=975 Fetches=0+0
~E#0007.01(1) BgrNdx1 Or__Map2  DBKeys=976 Fetches=0+0
~E#0007.01(1) BgrNdx2 FtchLim  DBKeys=1007  Fetches=1+9  RecsOut=1024
           1             1          1024
           1             1          1025
           1             1          1026
.
.
.
           1             1          1997
           1             1          1998
           1             1          1999
~E#0007.01(1) Fin     Bitmap   DBKeys=976  Fetches=0+11  RecsOut=2000
           1             1          2000
2000 rows selected
SQL>
```

First we observe delivery of the first 1024 rows using the fast first tactic. During this phase, background is not maintaining a dbkey list. When the 1025th dbkey is fetched, foreground is abandoned (ABA). The 1025th dbkey must be retained because it has not been delivered, so an in−memory BBC is constructed (Bld_Map) with 1 dbkey. The first 1024 dbkeys have been discarded.

The first background index is scanned to completion (EofData) accumulating a further 975 dbkeys. The dbkeys are sorted and rolled into a BBC (Bld_Map). The one dbkey from fast first termination is logically OR'ed with the 975 dbkeys from background (Or__Map) giving 976 dbkeys.

The second background index is scanned and reaches fetch limit (FtchLim) after 1007 dbkeys. Because this index is unique and less than 1024 dbkeys were read, these dbkeys would have been stored in a 1024 dbkey buffer. Since the index scan was aborted due to fetch limit, the dbkey buffer is discarded.

The final (Fin) phase delivers the remaining 976 rows using the background BBC from the first index scan.

This query shows an improvement of 71% in elapsed time and 78% in CPU time with the bitmap scan enhancements.

5.1.10 Enhancements to SQL SHOW Commands

This release of Oracle Rdb makes the following changes to the SHOW commands of Interactive SQL.

- New: SHOW SYSTEM TRIGGERS
  The SHOW SYSTEM TRIGGERS statement displays only those triggers created for use by the database system or layered applications such as the OCI Services component of SQL/Services.
- New: SHOW ALL TRIGGERS
  The SHOW ALL TRIGGERS statement displays both system and user defined triggers.
- New: SHOW SYSTEM MODULES
  The SHOW SYSTEM MODULES statement displays only those modules created for use by the database system or layered applications such as the OCI Services component of SQL/Services.
- New: SHOW ALL MODULES
  The SHOW ALL MODULES statement displays both system and user defined modules.
- New: SHOW SYSTEM FUNCTIONS
  The SHOW SYSTEM FUNCTIONS statement displays only those functions created for use by the database system or layered applications such as the OCI Services component of SQL/Services.
- New: SHOW ALL FUNCTIONS
  The SHOW ALL FUNCTIONS statement displays both system and user defined functions.
- New: SHOW SYSTEM PROCEDURES
  The SHOW SYSTEM PROCEDURES statement displays only those procedures created for use by the database system or layered applications such as the OCI Services component of SQL/Services.
- New: SHOW ALL PROCEDURES
  The SHOW ALL PROCEDURES statement displays both system and user defined procedures.
- SHOW now supports SQL wildcards in the object names
  Most SHOW commands allow a fully specified object name, or * to display details of all objects of the given type. With this release of Rdb, the LIKE wildcards "%" and "_" can be used to select a subset of object names. For instance, the following query will display all tables with the string "JOB" in the name.

  ```
  SQL> show table (comment) %JOB%
  Information for table CURRENT_JOB

  Comment on table CURRENT_JOB:
  View to provide the current job for employees

  Information for table JOBS

  Comment on table JOBS:
  Possible jobs in the company

  Information for table JOB_HISTORY

  Comment on table JOB_HISTORY:
  Employment history within the company

  SQL>
  ```

Note

This support is not currently available for multischema databases.

Refer to the documentation on the LIKE clause for information on the wildcard characters "%" and "_". The escape character is defined implicitly for SHOW commands as "\".

- SHOW allows synonyms to be used to identify the object to be displayed.
  The following example creates a sequence and a synonym for that sequence, and uses SHOW SEQUENCE with the synonym.

```
SQL> create sequence department_id_sequence;
SQL> create synonym dept_id_s for department_id_sequence;
SQL> show sequence
Sequences in database with filename personnel
     DEPARTMENT_ID_SEQUENCE
     DEPT_ID_S                   A synonym for sequence DEPARTMENT_ID_SEQUENCE
SQL> show sequence DEPT_ID_S
     DEPT_ID_S                   A synonym for sequence DEPARTMENT_ID_SEQUENCE
 Sequence Id: 1
 Initial Value: 1
 Minimum Value: 1
 Maximum Value: 9223372036854775787
 Next Sequence Value: 1
 Increment by: 1
 Cache Size: 20
 No Order
 No Cycle
 No Randomize
 Wait
SQL>
```

Note

This support is not currently available for multischema databases.

5.1.11 Return EXCESS_TRAN with Distributed Transaction

When Oracle Rdb is using distributed transactions and receives a SQL command to start a transaction while another transaction is in progress, it waits for the existing transaction to complete prior to starting the new transaction. This is necessary because of a race condition which can occur in DECdtm and which can result in Rdb returning an %RDB−E−EXCESS_TRANS error even though the prior transaction commit or rollback has been started.

An undesirable side effect of the wait described above is that if a second transaction is started without committing or rolling back the first (which is an error), Rdb will hang in an LEF wait state rather than returning an %RDB−E−EXCESS_TRANS error.

To deal with this problem, Oracle Rdb has been enhanced so that if a new distributed transaction is started, it checks to see if any current one has begun the two phase commit (2PC) or rollback processing. If so, it enters a wait state until the previous transaction is complete (as previously). But, if the transaction is not being ended, it pauses for a while to see if the transaction end processing begins and, if it does not, it returns an %RDB−E−EXCESS_TRANS error. A new configuration file parameter, called SQL_TRANS_START_WAIT, has been added to specify the number of seconds Rdb will wait for the end transaction processing before returning an %RDB−E−EXCESS_TRANS error. Rdb will pause in 500 millisecond intervals and recheck the existing transaction for up to the number of seconds specified by SQL_TRANS_START_WAIT, with a default of 3. SQL_TRANS_START_WAIT is defined in the client configuration file but may be specified for both the client and server sides of a remote connection.

The following example shows the new behavior of Oracle Rdb based on the setting of SQL_TRANS_START_WAIT. Suppose the RDB$CLIENT_DEFAULTS.DAT file contains the following entry:

```
SQL_TRANS_START_WAIT 4
```

Consider the following code fragment from a precompiled C source file which uses DECdtm system calls and explicitly passes distributed transaction context IDs to SQL:

```
struct {
    long version;
    long type;
    long length;
    char global_tid[16];
    long end;
    } context1, context2;
long status;
short iosb[4];
long flag = 2;

exec sql declare alias filename personnel;

context1.version = 1;
context1.type = 1;
context1.length = 16;
for ( i = 0; i < 16; i++ )
    context1.global_tid[i] = 0;
context1.end = 0;

context2.version = 1;
context2.type = 1;
context2.length = 16;
for ( i = 0; i < 16; i++ )
    context2.global_tid[i] = 0;
context2.end = 0;

status = sys$start_transw(
    0, /* efn */
    flag, /* flags */
    iosb, /* iosb */
    0, /* astadr */
    0, /* astprm */
    context1.global_tid /* tid */
    );

status = sys$start_transw(
    0, /* efn */
    flag, /* flags */
    iosb, /* iosb */
    0, /* astadr */
    0, /* astprm */
    context2.global_tid /* tid */
    );
```

```
exec sql using context :context1 set transaction read write;
```

```
exec sql using context :context2 set transaction read write;
```

In the above example, the second "set transaction" should result in an %RDB−E−EXCESS_TRANS error because Rdb does not allow more than one concurrent transaction. But, because a distributed transaction is involved, it would have hung in an LEF wait state. With the SQL_TRANS_START_WAIT parameter set to four seconds, it will check to see if the first transaction's end processing has begun every 500 milliseconds for four seconds and then return an %RDB−E−EXCESS_TRANS error.

5.1.12 Dynamic SQL Enhancements

The following enhancements have been made to the Rdb Dynamic SQL interface.

- The EXECUTE statement now supports INTO with a list of output host variables. In prior versions, only an INTO SQLDA was supported. See Section 5.1.8 for more details.
- DECLARE is now supported for creating local variables. Refer to the Oracle Rdb SQL Reference Manual, DECLARE Variable Statement. This statement is currently described as available in Interactive SQL only but is now available for dynamic SQL applications. These local variables will exist until a successful UNDECLARE or until the image runs down.
- UNDECLARE is now supported for deleting local variables. Refer to the Oracle Rdb SQL Reference Manual, UNDECLARE Variable Statement. This statement is currently described as available in Interactive SQL only but is now available for dynamic SQL applications.
- The PREPARE statement now sets values in the SQLCA to report the number of input and number output parameters for a statement. These values allow memory to be allocated for input and output SQLDA structures.
  Assuming that the SQLERRD array is zero based, SQL will set SQLERRD[2] to the count of output parameters and SQLERRD[3] to the count of input parameters. The values may possibly be zero and CALL parameters of INOUT type will appear in both the input and output count.
  The following simple program shows the effect of the new SQLCA support.

```
#include <stdio.h>
#include <sql_rdb_headers.h>

exec sql
    declare alias filename 'db$:mf_personnel';

exec sql
    include SQLCA;

char * s1 = "begin insert into work_status values (?, ?, ?);\
             select count(*) into ? from work_status; end";

void main ()
{
int i;
SQLCA.SQLERRD[2] = SQLCA.SQLERRD[3] = 1;
exec sql
    prepare stmt from :s1;
if (SQLCA.SQLCODE != 0) sql_signal ();
printf( "SQLCA:\n  SQLCODE:    %9d\n", SQLCA.SQLCODE);
for (i = 0; i < 6; i++)
    printf( "  SQLERRD[%d]: %9d\n", i, SQLCA.SQLERRD[i]);
}
```

The results below show that there are 3 input arguments and 1 output argument.

```
SQLCA:
  SQLCODE:             0
  SQLERRD[0]:          0
  SQLERRD[1]:          0
  SQLERRD[2]:          1
  SQLERRD[3]:          3
  SQLERRD[4]:          0
  SQLERRD[5]:          0
```

Please note that the SQLCA was not set prior to Rdb Release 7.1.3 so Oracle recommends that the SQLERRD[2] and SQLERRD[3] values be set to a known value (such as −1) prior to the PREPARE call. Then if the values remain as −1 the application must estimate the counts itself.

Chapter 6
Documentation Corrections, Additions and Changes

This chapter provides corrections for documentation errors and omissions.

6.1 Documentation Corrections

6.1.1 RDM$BIND_MAX_DBR_COUNT Documentation Clarification

Bugs 1495227 and 3916606

The Rdb7 Guide to Database Performance and Tuning Manual, Volume 2, page A−18, incorrectly describes the use of the RDM$BIND_MAX_DBR_COUNT logical.

Following is an updated description. Note that the difference in actual behavior between what is in the existing documentation and software is that the logical name only controls the number of database recovery processes created at once during "node failure" recovery (that is, after a system or monitor crash or other abnormal shutdown) for each database.

When an entire database is abnormally shut down (due, for example, to a system failure), the database will have to be recovered in a "node failure" recovery mode. This recovery will be performed by another monitor in the cluster if the database is opened on another node or will be performed the next time the database is opened.

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 for each database during a "node failure" recovery.

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 during a "node failure" recovery.

In a node failure recovery situation with the Row Cache feature enabled (regardless of the global buffer state), the database monitor will start a single database recovery (DBR) process to recover the Row Cache Server (RCS) process and all user processes from the oldest active checkpoint in the database.

Per−Database Value

The RDM$BIND_MAX_DBR_COUNT logical name specifies the maximum number of database recovery processes to run at once for each database. For example, if there are 10 databases being recovered and the value for the RDM$BIND_MAX_DBR_COUNT logical name is 8, up to 80 database recovery processes would be started by the monitor after a node failure.

The RDM$BIND_MAX_DBR_COUNT logical name is translated when the monitor process opens a database. Databases should be closed and reopened for a new value of the logical to become effective.

6.1.2 Database Server Process Priority Clarification

By default, the database servers (ABS, ALS, DBR, LCS, LRS, RCS) created by the Rdb monitor inherit their VMS process scheduling base priority from the Rdb monitor process. The default priority for the Rdb monitor process is 15.

Individual server priorities can be explicitly controlled via system−wide logical names as described in Table 6−1.

Table 6−1 Server Process Priority Logical Names

Logical Name	Use
RDM$BIND_ABS_PRIORITY	Base Priority for the ABS Server process
RDM$BIND_ALS_PRIORITY	Base Priority for the ALS Server process
RDM$BIND_DBR_PRIORITY	Base Priority for the DBR Server process
RDM$BIND_LCS_PRIORITY	Base Priority for the LCS Server process
RDM$BIND_LRS_PRIORITY	Base Priority for the LRS Server process
RDM$BIND_RCS_PRIORITY	Base Priority for the RCS Server process

When the Hot Standby feature is installed, the RDMAIJSERVER account is created specifying an account priority of 15. The priority of AIJ server processes on your system can be restricted with the system−wide logical name RDM$BIND_AIJSRV_PRIORITY. If this logical name is defined to a value less than 15, an AIJ server process will adjust its base priority to the value specified when the AIJ server process starts. Values from 0 to 31 are allowed for RDM$BIND_AIJSRV_PRIORITY, but the process is not able to raise its priority above the RDMAIJSERVER account value.

For most applications and systems, Oracle discourages changing the server process priorities.

6.1.3 Explanation of SQL$INT in a SQL Multiversion Environment and How to Redefine SQL$INT

Bug 2500594

In an environment running multiple versions of Oracle Rdb, for instance Rdb V7.0 and Rdb V7.1, there are now several varianted SQL images, such as SQL$70.EXE and SQL$71.EXE. However, SQL$INT.EXE is not varianted but acts as a dispatcher using the translation of the logical name RDMS$VERSION_VARIANT to activate the correct SQL runtime environment. This image is replaced when a higher version of Oracle Rdb is installed. Thus, using the example above, when Rdb V7.1 is installed, SQL$INT.EXE will be replaced with the V7.1 SQL$INT.EXE.

If an application is linked in this environment (using V7.1 SQL$INT) and the corresponding executable deployed to a system running Oracle Rdb V7.0 multiversion only, the execution of the application may result in the following error:

```
%IMGACT−F−SYMVECMIS, shareable image's symbol vector table mismatch
```

In order to avoid such a problem, the following alternative is suggested:

In the multiversion environment running both Oracle Rdb V7.0 and Oracle Rdb V7.1, run Oracle Rdb V7.0 multiversion by running the command procedures RDB$SETVER.COM 70 and RDB$SETVER RESET. This will set up the necessary logical names and symbols that establish the Oracle Rdb V7.0 environment.

For example:

```
$ @SYS$LIBRARY:RDB$SETVER 70

Current PROCESS Oracle Rdb environment is version V7.0−63 (MULTIVERSION)
Current PROCESS SQL environment is version V7.0−63 (MULTIVERSION)
Current PROCESS Rdb/Dispatch environment is version V7.0−63 (MULTIVERSION)

$ @SYS$LIBRARY:RDB$SERVER RESET
```

Now run SQL and verify that the version is correct:

```
$ sql$
SQL> show version
Current version of SQL is: Oracle Rdb SQL V7.0−63
```

Define SQL$INT to point to the varianted SQL$SHR.EXE. Then, create an options file directing the linker to link with this newly defined SQL$INT. An example follows:

```
$ DEFINE SQL$INT SYS$SHARE:SQL$SHR'RDMS$VERSION_VARIANT'.EXE
$ LINK TEST_APPL,SQL$USER/LIB,SYS$INPUT/option
SQL$INT/SHARE
^Z
```

The executable is now ready to be deployed to the Oracle Rdb V7.0 multiversion environment and should run successfully.

Please note that with each release of Oracle Rdb, new entry points are added to the SQL$INT shareable image. This allows the implementation of new functionality. Therefore, applications linked with SQL$INT from Oracle Rdb V7.1 cannot be run on systems with only Oracle Rdb V7.0 installed. This is because the shareable image does not contain sufficient entry points.

The workaround presented here allows an application to explicitly link with the Oracle Rdb V7.0 version of the image. Such applications are upward compatible and will run on Oracle Rdb V7.0 and Oracle Rdb V7.1. The applications should be compiled and linked under the lowest version.

In environments where Oracle Rdb V7.1 is installed, this workaround is not required because the SQL$INT image will dynamically activate the appropriate SQL$SHRxx image as expected.

6.1.4 Documentation Omitted Several Reserved Words

Bug 2319321

The following keywords are considered reserved words in Oracle Rdb Release 7.1.

- UID
- CURRENT_UID
- SYSTEM_UID
- SESSION_UID
- RAW

- LONG
- DBKEY
- ROWID
- SYSDATE

In particular, any column which has these names will be occluded by the keyword. i.e. selecting from column UID will be interpreted as referencing the built in function UID and so return a different result.

The correction to this problem is to enable keyword quoting using SET QUOTING RULES 'SQL92' (or 'SQL99') and enclose the column name in quotations.

In addition, SQL will now generate a warning if these reserved words are used (unquoted) in CREATE and ALTER operations.

6.1.5 Using Databases from Releases Earlier Than V6.0

Bug 2383967

You cannot convert or restore databases earlier than V6.0 directly to V7.1. The RMU Convert command for V7.1 supports conversions from V6.0 through V7.0 only. If you have a V3.0 through V5.1 database, you must convert it to at least V6.0 and then convert it to V7.1. For example, if you have a V4.2 database, convert it first to at least V6.0, then convert the resulting database to V7.1.

If you attempt to convert a database created prior to V6.0 directly to V7.1, Oracle RMU generates an error.

6.1.6 New RMU/BACKUP Storage Area Assignment With Thread Pools

This is to clarify how storage areas are assigned to disk and tape devices using the new RMU/BACKUP THREAD POOL and BACKUP TO MULTIPLE DISK DEVICES features introduced in Oracle Rdb Release 7.1.

For the case of backup to multiple disk devices using thread pools, the algorithm used by RMU/BACKUP to assign threads is to calculate the size of each area as the product of the page length in bytes times the highest page number used (maximum page number) for that area. The area sizes are then sorted by descending size and ascending device name. For internal processing reasons, the system area is placed as the first area in the first thread. Each of the remaining areas is added to whichever thread has the lowest byte count. In this way, the calculated area sizes are balanced between the threads.

For tape devices, the same algorithm is used but the areas are partitioned among writer threads, not disk devices.

The partitioning for backup to multiple disk devices is done by disk device, not by output thread, because there will typically be more disk devices than output threads, and an area can not span a device.

6.1.7 RDM$BIND_LOCK_TIMEOUT_INTERVAL Overrides the Database Parameter

Bug 2203700

When starting a transaction, there are three different values that are used to determine the lock timeout interval for that transaction. Those values are:

1. The value specified in the SET TRANSACTION statement
2. The value stored in the database as specified in CREATE or ALTER DATABASE
3. The value of the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL

The timeout interval for a transaction is the smaller of the value specified in the SET TRANSACTION statement and the value specified in CREATE DATABASE. However, if the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL is defined, the value of this logical name overrides the value specified in CREATE DATABASE.

The description of how these three values interact, found in several different parts of the Rdb documentation set, is incorrect and will be replaced by the description above.

The lock timeout value in the database can be dynamically modified from the Locking Dashboard in RMU/SHOW STATISTICS. The Per−Process Locking Dashboard can be used to dynamically override the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL for one or more processes.

6.1.8 New Request Options for RDO, RDBPRE and RDB$INTERPRET

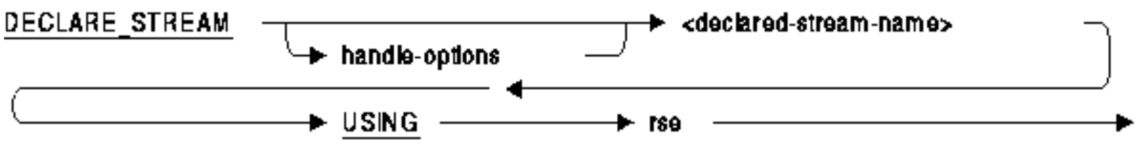
This release note was included in the V70A Release Notes but had gotten dropped somewhere along the line.

For this release of Rdb, two new keywords have been added to the handle−options for the DECLARE_STREAM, the START_STREAM (undeclared format) and FOR loop statements. These changes have been made to RDBPRE, RDO and RDB$INTERPRET at the request of several RDO customers.

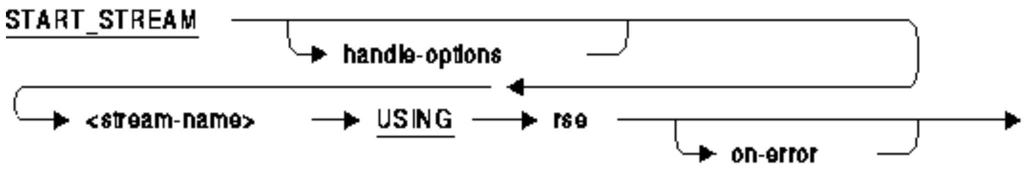
In prior releases, the handle−options could not be specified in interactive RDO or RDB$INTERPRET. This has changed in Rdb7 but these allowed options will be limited to MODIFY and PROTECTED keywords. For RDBPRE, all options listed will be supported. These option names were chosen to be existing keywords to avoid adding any new keywords to the RDO language.

The altered statements are shown in Example 5−1, Example 5−2 and Example 5−3.

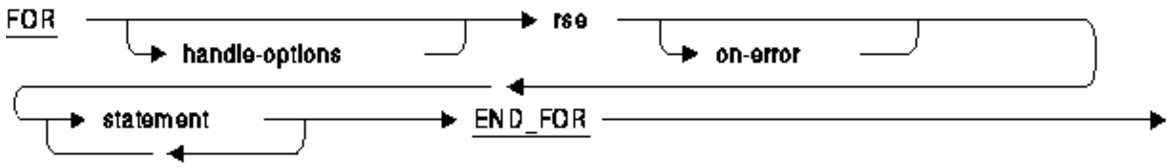
Example 5−1 DECLARE_STREAM Format



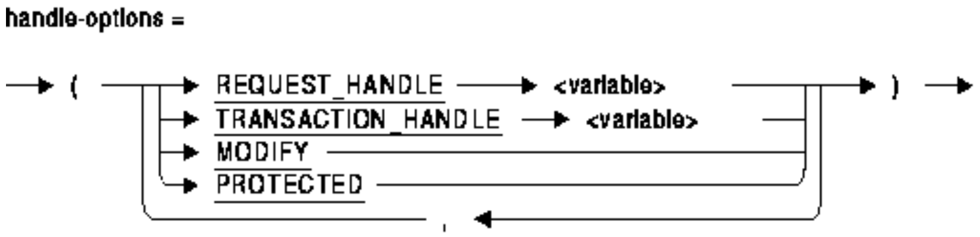
Example 5−2 START_STREAM Format



Example 5−3 FOR Format



Each of these statements references the syntax for the HANDLE−OPTIONS which has been revised and is shown below.



The following options are available for HANDLE−OPTIONS:

- REQUEST_HANDLE specifies the request handle for this request. This option is only valid for RDBPRE and RDML applications. It cannot be used with RDB$INTERPRET, nor interactive RDO.
- TRANSACTION_HANDLE specifies the transaction handle under which this request executes. This option is only valid for RDBPRE and RDML applications. It cannot be used with RDB$INTERPRET, nor interactive RDO.
- MODIFY specifies that the application will modify all (or most) records fetched from the stream or for loop. This option can be used to improve application performance by avoiding lock promotion from SHARED READ for the FETCH to PROTECTED WRITE access for the nested MODIFY or ERASE statement. It can also reduce DEADLOCK occurrence because lock promotions are avoided. This option is valid for RDBPRE, RDB$INTERPRET, and interactive RDO. This option is not currently available for RDML.
  For example:

```
RDO>   FOR (MODIFY) E IN EMPLOYEES WITH E.EMPLOYEE_ID = "00164"
cont>    MODIFY E USING E.MIDDLE_INITIAL = "M"
cont>    END_MODIFY
cont>  END_FOR
```

This FOR loop uses the MODIFY option to indicate that the nested MODIFY is an unconditional statement and so aggressive locking can be undertaken during the fetch of the record in the FOR loop.

- PROTECTED specifies that the application may modify records fetched by this stream by a separate and independent MODIFY statement. Therefore, this stream should be protected from interference (aka Halloween affect). The optimizer will select a snapshot of the rows and store them in a temporary relation for processing, rather than traversing indexes at the time of the FETCH statement. In some cases this may result in poorer performance when the temporary relation is large and overflows from virtual memory to a temporary disk file, but the record stream will be protected from interference. The programmer is directed to the documentation for the Oracle Rdb logical names RDMS$BIND_WORK_VM and RDMS$BIND_WORK_FILE.
  This option is valid for RDBPRE, RDB$INTERPRET, and interactive RDO. This option is not currently available for RDML.
  The following example creates a record stream in a BASIC program using Callable RDO:

```
RDMS_STATUS = RDB$INTERPRET ('INVOKE DATABASE PATHNAME "PERSONNEL"')

RDMS_STATUS = RDB$INTERPRET ('START_STREAM (PROTECTED) EMP USING ' + &
                             'E IN EMPLOYEES')

RDMS_STATUS = RDB$INTERPRET ('FETCH EMP')

DML_STRING = 'GET ' +                                         &
                 '!VAL = E.EMPLOYEE_ID;' +                    &
                 '!VAL = E.LAST_NAME;' +                      &
                 '!VAL = E.FIRST_NAME' +                      &
             'END_GET'

RDMS_STATUS = RDB$INTERPRET (DML_STRING, EMP_ID, LAST_NAME, FIRST_NAME)
```

  In this case the FETCH needs to be protected against MODIFY statements which execute in other parts of the application.

6.2 Address and Phone Number Correction for Documentation

In release 7.0 or earlier documentation, the address and fax phone number listed on the Send Us Your Comments page are incorrect. The correct information is:

```
FAX -- 603.897.3825
Oracle Corporation
One Oracle Drive
Nashua, NH 03062-2804
USA
```

6.3 Online Document Format and Ordering Information

You can view the documentation in Adobe Acrobat format using the Acrobat Reader, which allows anyone to view, navigate, and print documents in the Adobe Portable Document Format (PDF). See http://www.adobe.com for information about obtaining a free copy of Acrobat Reader and for information on supported platforms.

The Oracle Rdb documentation in Adobe Acrobat format is available on MetaLink:

```
Top Tech Docs\Oracle Rdb\Documentation\<bookname>
```

Customers should contact their Oracle representative to purchase printed documentation.

6.4 New and Changed Features in Oracle Rdb Release 7.1

This section provides information about late−breaking new features or information that is missing or changed since the Oracle Rdb New and Changed Features for Oracle Rdb manual was published.

6.4.1 PERSONA is Supported in Oracle SQL/Services

In the "New and Changed Features for Oracle Rdb" Manual under the section "ALTER DATABASE Statement" is a note stating that impersonation is not supported in Oracle SQL/Services. This is incorrect. There was a problem in the first release of Oracle Rdb 7.1 (7.1.0) whereby impersonation through Oracle SQL/Services failed. This problem is resolved in Oracle Rdb Release 7.1.0.1.

6.4.2 NEXTVAL and CURRVAL Pseudocolumns Can Be Delimited Identifiers

The New and Changed Features for Oracle Rdb manual describes SEQUENCES but does not mention that the special pseudocolumns NEXTVAL and CURRVAL can be delimited. All uppercase and lowercase variations of these keywords are accepted and assumed to be equivalent to these uppercase keywords.

The following example shows that any case is accepted:

```
SQL> set dialect 'sql92';
SQL> create sequence dept_id;
SQL> select dept_id.nextval from rdb$database;
                    1
1 row selected
SQL> select "DEPT_ID".currval from rdb$database;
                    1
1 row selected
SQL> select "DEPT_ID"."CURRVAL" from rdb$database;
                    1
1 row selected
SQL> select "DEPT_ID"."nextval" from rdb$database;
                    2
1 row selected
SQL> select "DEPT_ID"."CuRrVaL" from rdb$database;
                    2
1 row selected
```

6.4.3 Only=select_list Qualifier for the RMU Dump After_Journal Command

The Oracle Rdb New and Changed Features for Oracle Rdb manual documents the First=select_list and Last=select_list qualifiers for the RMU Dump After_Journal command. Inadvertently missed was the Only=select_list qualifier.

The First, Last, and Only qualifiers have been added because the Start and End qualifiers are difficult to use since users seldom know, nor can they determine, the AIJ record number in advance of using the RMU Dump

After_Journal command.

The select_list clause of these qualifiers consists of a list of one or more of the following keywords:

- TSN=tsn
  Specifies the first, last, or specific TSN in the AIJ journal using the standard [n:]m TSN format.
- TID=tid
  Specifies the first, last or specific TID in the AIJ journal.
- RECORD=record
  Specifies the first or last record in the AIJ journal. This is the same as the existing Start and End qualifiers (which are still supported, but deprecated). This keyword cannot be used with the Only qualifier.
- BLOCK=block#
  Specifies the first or last block in the AIJ journal. This keyword cannot be used with the Only qualifier.
- TIME=date_time
  Specifies the first or last date/time in the AIJ journal using the standard date/time format. This keyword cannot be used with the Only qualifier.

The First, Last, and Only qualifiers are optional. You may specify any or none of them.

The keywords specified for the First qualifier can differ from the keywords specified for the other qualifiers.

For example, to start the dump from the fifth block of the AIJ journal, you would use the following command:

```
RMU/DUMP/AFTER_JOURNAL /FIRST=(BLOCK=5) MF_PERSONNEL.AIJ
```

To start the dump from block 100 or TSN 52, whichever occurs first, you would use the following command:

```
RMU/DUMP/AFTER_JOURNAL /FIRST=(BLOCK=100,TSN=0:52) MF_PERSONNEL.AIJ
```

When multiple keywords are specified for a qualifier, the first condition being encountered activates the qualifier. In the preceding example, the dump starts when either block 100 or TSN 52 is encountered.

Be careful when searching for TSNs or TIDs as they are not ordered in the AIJ journal. For example, if you want to search for a specific TSN, use the Only qualifier and not the First and Last qualifiers. For example, assume the AIJ journal contains records for TSN 150, 170, and 160 (in that order). If you specify the First=TSN=160 and Last=TSN=160 qualifiers, nothing will be dumped because TSN 170 will match the Last=TSN=160 criteria.

6.5 Oracle Rdb7 and Oracle CODASYL DBMS Guide to Hot Standby Databases

This section provides information that is missing from or changed in V7.0 of the Oracle Rdb7 and Oracle CODASYL DBMS Guide to Hot Standby Databases.

6.5.1 Restrictions Lifted on After−Image Journal Files

The Hot Standby software has been enhanced regarding how it handles after−image journal files. Section 4.2.4 in the Oracle Rdb and Oracle CODASYL DBMS Guide to Hot Standby Databases states the following information:

```
If an after−image journal switchover operation is suspended when
replication operations are occurring, you must back up one or more of
the modified after−image journals to add a new journal file.
```

This restriction has been removed. Now, you can add journal files or use the emergency AIJ feature of Oracle Rdb release 7.0 to automatically add a new journal file. Note the following distinctions between adding an AIJ file and adding an emergency AIJ file:

- You can add an AIJ file to the master database and it will be replicated on the standby database. If replication operations are active, the AIJ file is created on the standby database immediately. If replication operations are not active, the AIJ file is created on the standby database when replication operations are restarted.
- You can add emergency AIJ files anytime. If replication operations are active, the emergency AIJ file is created on the standby database immediately. However, because emergency AIJ files are not journaled, starting replication after you create an emergency AIJ will fail. You cannot start replication operations because the Hot Standby software detects a mismatch in the number of after−image journal files on the master compared to the standby database.
  If an emergency AIJ file is created on the master database when replication operations are not active, you must perform a master database backup and then restore the backup on the standby database. Otherwise, an AIJSIGNATURE error results.

6.5.2 Changes to RMU Replicate After_Journal ... Buffer Command

The behavior of the RMU Replicate After_Journal ... Buffers command has been changed. The Buffers qualifier may be used with either the Configure option or the Start option.

When using local buffers, the AIJ Log Roll−forward Server will use a minimum of 4096 buffers. The value provided to the Buffers qualifier will be accepted but ignored if it is less than 4096. In addition, further parameters will be checked and the number of buffers may be increased if the resulting calculations are greater than the number of buffers specified by the Buffers qualifier. If the database is configured to use more than 4096 AIJ Request Blocks (ARBs), then the number of buffers may be increased to the number of ARBs configured for the database. The LRS ensures that there are at least 10 buffers for every possible storage area in the database. Thus if the total number of storage areas (both used and reserved) multiplied by 10 results in a greater number of buffers, then that number will be used.

When global buffers are used, the number of buffers used by the AIJ Log Roll−forward Server is determined as follows:

- If the Buffers qualifier is omitted and the Online qualifier is specified, then the number of buffers will default to the previously configured value, if any, or 256, whichever is larger.
- If the Buffers qualifier is omitted and the Online qualifier is not specified or the Noonline qualifier is specified, then the number of buffers will default to the maximum number of global buffers allowed per user ("USER LIMIT"), or 256, whichever is larger.
- If the Buffers qualifier is specified then that value must be at least 256, and it may not be greater than the maximum number of global buffers allowed per user ("USER LIMIT").

The Buffer qualifier now enforces a minimum of 256 buffers for the AIJ Log Roll−forward Server. The maximum number of buffers allowed is still 524288 buffers.

6.5.3 Unnecessary Command in the Hot Standby Documentation

There is an unnecessary command documented in the Oracle Rdb and Oracle CODASYL DBMS Guide to Hot Standby Databases manual. The documentation (in Section 2.12 "Step 10: Specify the Network Transport Protocol") says that to use TCP/IP as the network protocol, you must issue the following commands:

```
$ CONFIG UCX AIJSERVER OBJECT
$ UCX SET SERVICE RDMAIJSRV
/PORT=n
/USER_NAME=RDMAIJSERVER
/PROCESS_NAME=RDMAIJSERVER
/FILE=SYS$SYSTEM:rdmaijserver_ucx.com
/LIMIT=nn
```

The first of these commands ($ CONFIG UCX AIJSERVER OBJECT) is unnecessary. You can safely disregard the first line when setting up to use TCP/IP with Hot Standby.

The documentation will be corrected in a future release of Oracle Rdb.

6.5.4 Change in the Way RDMAIJ Server is Set Up in UCX

Starting with Oracle Rdb Release 7.0.2.1, the RDMAIJ image became a varianted image. Therefore, the information in Section 2.12, "Step 10: Specify the Network Transport Protocol," of the Oracle Rdb7 and Oracle CODASYL DBMS Guide to Hot Standby Databases has become outdated with regard to setting up the RDMAIJSERVER object when using UCX as the network transport protocol. The UCX SET SERVICE command is now similar to the following:

```
$ UCX SET SERVICE RDMAIJ −
      /PORT=port_number −
      /USER_NAME=RDMAIJ −
      /PROCESS_NAME=RDMAIJ −
      /FILE=SYS$SYSTEM:RDMAIJSERVER.com −
      /LIMIT=limit
```

For Oracle Rdb multiversion, the UCX SET SERVICE command is similar to the following:

```
$ UCX SET SERVICE RDMAIJ70 −
      /PORT=port_number −
      /USER_NAME=RDMAIJ70 −
      /PROCESS_NAME=RDMAIJ70 -
      /FILE=SYS$SYSTEM:RDMAIJSERVER70.com -
      /LIMIT=limit
```

The installation procedure for Oracle Rdb creates a user named RDMAIJ(nn) and places a file called RDMAIJSERVER(nn).COM in SYS$SYSTEM. The RMONSTART(nn).COM command procedure will try to enable a service called RDMAIJ(nn) if UCX is installed and running.

Changing the RDMAIJ server to a varianted image does not impact installations using DECNet since the correct DECNet object is created during the Oracle Rdb installation.

6.5.5 CREATE INDEX Operation Supported for Hot Standby

On Page 1−13 of the Oracle Rdb7 and Oracle CODASYL DBMS Guide to Hot Standby Databases, the add new index operation is incorrectly listed as an offline operation not supported by Hot Standby. The CREATE INDEX operation is now fully supported by Hot Standby, as long as the transaction does not span all available AIJ journals, including emergency AIJ journals.

6.6 Oracle Rdb7 for OpenVMS Installation and Configuration Guide

This section provides information that is missing from or changed in V7.0 of the Oracle Rdb7 for OpenVMS Installation and Configuration Guide.

6.6.1 Suggestion to Increase GH_RSRVPGCNT Removed

The Oracle Rdb7 for OpenVMS Installation and Configuration Guide contains a section titled "Installing Oracle Rdb Images as Resident on OpenVMS Alpha". This section includes information about increasing the value of the OpenVMS system parameter GH_RSRVPGCNT when you modify the RMONSTART.COM or SQL$STARTUP.COM procedures to install Oracle Rdb images with the Resident qualifier.

Note that modifying the parameter GH_RSRVPGCNT is only required if the RMONSTART.COM or SQL$STARTUP.COM procedures have been manually modified to install Oracle Rdb images with the Resident qualifier. Furthermore, if the RMONSTART.COM and SQL$STARTUP.COM procedures are executed during the system startup procedure (directly from SYSTARTUP_VMS.COM, for example), there is no need to modify the GH_RSRVPGCNT parameter.

Oracle Corporation recommends that you do not modify the value of the GH_RSRVPGCNT system parameter unless it is absolutely required. Some versions of OpenVMS on some hardware platforms require GH_RSRVPGCNT to be a value of zero in order to ensure the highest level of system performance.

6.6.2 Prerequisite Software

In addition to the software listed in the Oracle Rdb Installation and Configuration Guide and at the url http://www.oracle.com/rdb/product_info/index.html, note that the MACRO−32 compiler and the OpenVMS linker are required OpenVMS components in order to install Oracle Rdb on your OpenVMS Alpha system.

The MACRO−32 Compiler for OpenVMS Alpha is a standard component of the OpenVMS Operating System. It is used to compile :VAX MACRO assembly language source files into native OpenVMS Alpha object code. During the Oracle Rdb installation procedure, and portions of the installation verification procedure (such as the test for RDBPRE), the MACRO−32 compiler is required.

The OpenVMS linker is a standard component of the OpenVMS Operating System. It is used to link one or more input files into a program image and defines the execution characteristics of the image. The linker will be required for application development and is likewise used by the Oracle Rdb installation procedure and the installation verification procedure.

6.6.3 Defining the RDBSERVER Logical Name

Sections 4.3.7.1 and 4.3.7.2 in the Oracle Rdb7 for OpenVMS Installation and Configuration Guide provide the following examples for defining the RDBSERVER logical name: *$ DEFINE RDBSERVER SYS$SYSTEM:RDBSERVER70.EXE*

and *$ DEFINE RDBSERVER SYS$SYSTEM:RDBSERVER61.EXE*

These definitions are inconsistent with other command procedures that attempt to reference the RDBSERVERxx.EXE image. Below is one example where the RDBSERVER.COM procedure references SYS$COMMON:<SYSEXE> and SYS$COMMON:[SYSEXE] rather than SYS$SYSTEM.

```
$   if .not. -
       ((f$locate ("SYS$COMMON:<SYSEXE>",rdbserver_image) .ne. log_len) .or. -
        (f$locate ("SYS$COMMON:[SYSEXE]",rdbserver_image) .ne. log_len))
$   then
$       say "''rdbserver_image' is not found in SYS$COMMON:<SYSEXE>"
$       say "RDBSERVER logical is ''rdbserver_image'"
$       exit
$   endif
```

In this case, if the logical name were defined as instructed in the Oracle Rdb7 for OpenVMS Installation and Configuration Guide, the image would not be found.

The correct definition of the logical name is as follows: *DEFINE RDBSERVER SYS$COMMON:<SYSEXE>RDBSERVER70.EXE*

and *DEFINE RDBSERVER SYS$COMMON:<SYSEXE>RDBSERVER61.EXE*

6.7 Guide to Database Design and Definition

This section provides information that is missing from or changed in release 7.0 of the Oracle Rdb7 Guide to Database Design and Definition.

6.7.1 Lock Timeout Interval Logical Incorrect

On Page 7−31 of Section 7.4.8 in the Oracle Rdb7 Guide to Database Design and Definition, the RDM$BIND_LOCK_TIMEOUT logical name is referenced incorrectly. The correct logical name is RDM$BIND_LOCK_TIMEOUT_INTERVAL.

The Oracle Rdb7 Guide to Database Design and Definition will be corrected in a future release.

6.7.2 Example 4−13 and Example 4−14 Are Incorrect

Example 4−13 showing vertical partitioning, and Example 4−14, showing vertical and horizontal partitioning, are incorrect. They should appear as follows:

Example 4−13:

```
SQL> CREATE STORAGE MAP EMPLOYEES_1_MAP
cont>    FOR EMPLOYEES
cont>    ENABLE COMPRESSION
cont>    STORE COLUMNS (EMPLOYEE_ID, LAST_NAME, FIRST_NAME,
cont>                   MIDDLE_INITIAL, STATUS_CODE)
cont>                   DISABLE COMPRESSION
cont>                   IN ACTIVE_AREA
cont>    STORE COLUMNS (ADDRESS_DATA_1, ADDRESS_DATA_2, CITY,
cont>                   STATE, POSTAL_CODE)
cont>                   IN INACTIVE_AREA
cont>    STORE IN OTHER_AREA;
```

Example 4−14:

```
SQL>  CREATE STORAGE MAP EMPLOYEES_1_MAP2
cont>        FOR EMP2
cont>        STORE COLUMNS (EMPLOYEE_ID, LAST_NAME, FIRST_NAME,
cont>                       MIDDLE_INITIAL, STATUS_CODE)
cont>              USING (EMPLOYEE_ID)
cont>              IN ACTIVE_AREA_A WITH LIMIT OF ('00399')
cont>              IN ACTIVE_AREA_B WITH LIMIT OF ('00699')
cont>              OTHERWISE IN ACTIVE_AREA_C
cont>        STORE COLUMNS (ADDRESS_DATA_1, ADDRESS_DATA_2, CITY,
cont>                      STATE, POSTAL_CODE)
cont>              USING (EMPLOYEE_ID)
cont>              IN INACTIVE_AREA_A WITH LIMIT OF ('00399')
cont>              IN INACTIVE_AREA_B WITH LIMIT OF ('00699')
cont>              OTHERWISE IN INACTIVE_AREA_C
cont>        STORE IN OTHER_AREA;
```

6.8 Oracle RMU Reference Manual, Release 7.0

This section provides information that is missing from or changed in V7.0 of the Oracle RMU Reference Manual.

6.8.1 RMU Unload After_Journal Null Bit Vector Clarification

Each output record from the RMU /UNLOAD /AFTER_JOURNAL command includes a vector (array) of bits. There is one bit for each field in the data record. If a null bit value is 1, the corresponding field is NULL; if a null bit value is 0, the corresponding field is not NULL and contains an actual data value. The contents of a data field that is NULL are not initialized and are not predictable.

The null bit vector begins on a byte boundary. The field RDB$LM_NBV_LEN indicates the number of valid bits (and thus, the number of columns in the table). Any extra bits in the final byte of the vector after the final null bit are unused and the contents are unpredictable.

The following example C program demonstrates one possible way of reading and parsing a binary output file (including the null bit vector) from the RMU /UNLOAD /AFTER_JOURNAL command. This sample program has been tested using Oracle Rdb V7.0.5 and higher and HP C V6.2−009 on OpenVMS Alpha V7.2−1. It is meant to be used as a template for writing your own program.

```
/* DATATYPES.C */

#include <stdio.h>
#include <descrip.h>
#include <starlet.h>
#include <string.h>

#pragma member_alignment __save
#pragma nomember_alignment

struct { /* Database key structure */
    unsigned short      lno;    /* line number */
    unsigned int        pno;    /* page number */
    unsigned short      dbid;   /* area number */
    } dbkey;

typedef struct { /* Null bit vector with one bit for each column */
    unsigned            n_tinyint   :1;
    unsigned            n_smallint  :1;
    unsigned            n_integer   :1;
    unsigned            n_bigint    :1;
    unsigned            n_double    :1;
    unsigned            n_real      :1;
    unsigned            n_fixstr    :1;
    unsigned            n_varstr    :1;
    } nbv_t;

struct { /* LogMiner output record structure for table DATATYPES */
    char                rdb$lm_action;
    char                rdb$lm_relation_name [31];
    int                 rdb$lm_record_type;
    short               rdb$lm_data_len;
    short               rdb$lm_nbv_len;
    __int64             rdb$lm_dbk;
    __int64             rdb$lm_start_tad;
    __int64             rdb$lm_commit_tad;
    __int64             rdb$lm_tsn;
    short               rdb$lm_record_version;
    char                f_tinyint;
    short               f_smallint;
    int                 f_integer;
    __int64             f_bigint;
    double              f_double;
    float               f_real;
    char                f_fixstr[10];
    short               f_varstr_len;   /* length of varchar */
    char                f_varstr[10];   /* data of varchar */
    nbv_t               nbv;
    } lm;

#pragma member_alignment __restore

main ()
{   char timbuf [24];
    struct dsc$descriptor_s dsc = {
        23, DSC$K_DTYPE_T, DSC$K_CLASS_S, timbuf};
    FILE *fp = fopen ("datatypes.dat", "r", "ctx=bin");

    memset (&timbuf, 0, sizeof(timbuf));

    while (fread (&lm, sizeof(lm), 1, fp) != 0)
    {
        printf ("Action     = %c\n",     lm.rdb$lm_action);
        printf ("Table      = %.*s\n",   sizeof(lm.rdb$lm_relation_name),
                                         lm.rdb$lm_relation_name);
        printf ("Type       = %d\n",     lm.rdb$lm_record_type);
        printf ("Data Len   = %d\n",     lm.rdb$lm_data_len);
        printf ("Null Bits  = %d\n",     lm.rdb$lm_nbv_len);

        memcpy (&dbkey, &lm.rdb$lm_dbk, sizeof(lm.rdb$lm_dbk));
        printf ("DBKEY      = %d:%d:%d\n", dbkey.dbid,
                                        dbkey.pno,
                                        dbkey.lno);

        sys$asctim (0, &dsc, &lm.rdb$lm_start_tad, 0);
        printf ("Start TAD  = %s\n", timbuf);

        sys$asctim (0, &dsc, &lm.rdb$lm_commit_tad, 0);
        printf ("Commit TAD = %s\n", timbuf);

        printf ("TSN        = %Ld\n",    lm.rdb$lm_tsn);
        printf ("Version    = %d\n",     lm.rdb$lm_record_version);

        if (lm.nbv.n_tinyint == 0)
                printf ("f_tinyint  = %d\n", lm.f_tinyint);
        else    printf ("f_tinyint  = NULL\n");

        if (lm.nbv.n_smallint == 0)
                printf ("f_smallint = %d\n", lm.f_smallint);
        else    printf ("f_smallint = NULL\n");

        if (lm.nbv.n_integer == 0)
                printf ("f_integer  = %d\n", lm.f_integer);
        else    printf ("f_integer  = NULL\n");

        if (lm.nbv.n_bigint == 0)
```

```
                printf ("f_bigint   = %Ld\n", lm.f_bigint);
        else    printf ("f_bigint   = NULL\n");

        if (lm.nbv.n_double == 0)
                printf ("f_double   = %f\n", lm.f_double);
        else    printf ("f_double   = NULL\n");

        if (lm.nbv.n_real == 0)
                printf ("f_real     = %f\n", lm.f_real);
        else    printf ("f_real     = NULL\n");

        if (lm.nbv.n_fixstr == 0)
                printf ("f_fixstr   = %.*s\n", sizeof (lm.f_fixstr),
                                                        lm.f_fixstr);
        else    printf ("f_fixstr   = NULL\n");

        if (lm.nbv.n_varstr == 0)
                printf ("f_varstr   = %.*s\n", lm.f_varstr_len, lm.f_varstr);
        else    printf ("f_varstr   = NULL\n");

        printf ("\n");
    }
}
```

Example sequence of commands to create a table, unload the data and display the contents with this program:

```
SQL> ATTACH 'FILE MF_PERSONNEL';
SQL> CREATE TABLE DATATYPES (
       F_TINYINT TINYINT
      ,F_SMALLINT SMALLINT
      ,F_INTEGER INTEGER
      ,F_BIGINT BIGINT
      ,F_DOUBLE DOUBLE PRECISION
      ,F_REAL REAL
      ,F_FIXSTR CHAR (10)
      ,F_VARSTR VARCHAR (10));
SQL> COMMIT;
SQL> INSERT INTO DATATYPES VALUES (1, NULL, 2, NULL, 3, NULL, 'THIS', NULL);
SQL> INSERT INTO DATATYPES VALUES (NULL, 4, NULL, 5, NULL, 6, NULL, 'THAT');
SQL> COMMIT;
SQL> EXIT;
$ RMU /BACKUP /AFTER_JOURNAL MF_PERSONNEL AIJBCK.AIJ
$ RMU /UNLOAD /AFTER_JOURNAL MF_PERSONNEL AIJBCK.AIJ -
     /TABLE = (NAME=DATATYPES, OUTPUT=DATATYPES.DAT)
$ CC DATATYPES.C
$ LINK DATATYPES.OBJ
$ RUN DATATYPES.EXE
```

6.8.2 New Transaction_Mode Qualifier for Oracle RMU Commands

A new qualifier, Transaction_Mode, has been added to the RMU Copy, Move_Area, Restore, and Restore Only_Root commands. You can use this qualifier to set the allowable transaction modes for the database root file created by these commands. If you are not creating a root file as part of one of these commands, for example, you are restoring an area, attempting to use this qualifier returns a CONFLSWIT error. This qualifier is similar to the SET TRANSACTION MODE clause of the CREATE DATABASE command in interactive SQL.

The primary use of this qualifier is when you restore a backup file (of the master database) to create a Hot Standby database. Include the Transaction_Mode qualifier on the RMU Restore command when you create the standby database (prior to starting replication operations). Because only read−only transactions are allowed on the standby database, you should use the Transaction_Mode=Read_Only qualifier setting. This setting prevents modifications to the standby database at all times, even when replication operations are not active.

You can specify the following transaction modes for the Transaction_Mode qualifier:

```
All
Current
None
[No]Batch_Update
[No]Read_Only
[No]Exclusive
[No]Exclusive_Read
[No]Exclusive_Write
[No]Protected
[No]Protected_Read
[No]Protected_Write
[No]Shared
[No]Shared_Read
[No]Shared_Write
```

Note that [No] indicates that the value can be negated. For example, the NoExclusive_Write option indicates that exclusive write is not an allowable access mode for this database. If you specify the Shared, Exclusive, or Protected option, Oracle RMU assumes you are referring to both reading and writing in these modes. For example, the Transaction_Mode=Shared option indicates that you want both Shared_Read and Shared_Write as transaction modes. No mode is enabled unless you add that mode to the list or you use the ALL option to enable all modes.

You cannot negate the following three options: All, which enables all transaction modes; None, which disables all transaction modes; and Current, which enables all transaction modes that are set for the source database. If you do not specify the Transaction_Mode qualifier, Oracle RMU uses the transaction modes enabled for the source database.

You can list one qualifier that enables or disables a particular mode followed by another that does the opposite. For example, Transaction_Mode=(NoShared_Write, Shared) is ambiguous because the first value disables Shared_Write access while the second value enables Shared_Write access. Oracle RMU resolves the ambiguities by first enabling all modes that are enabled by the items in the Transaction_Mode list and then disabling those modes that are disabled by items in the Transaction_Mode list. The order of items in the list is irrelevant. In the example discussed, Shared_Read is enabled and Shared_Write is disabled.

The following example shows how to set a newly restored database to allow read−only transactions only. After Oracle RMU executes the command, the database is ready for you to start Hot Standby replication operations.

```
$ RMU/RESTORE/TRANSACTION_MODE=READ_ONLY MF_PERSONNEL.RBF
```

6.8.3 RMU Server After_Journal Stop Command

If database replication is active and you attempt to stop the database AIJ Log Server, Oracle Rdb returns an error. You must stop database replication before attempting to stop the server.

In addition, a new qualifier, Output=filename, has been added to the RMU Server After_Journal Stop command. This optional qualifier allows you to specify the file where the operational log is to be created. The operational log records the transmission and receipt of network messages.

If you do not include a directory specification with the file name, the log file is created in the database root file directory. It is invalid to include a node name as part of the file name specification.

Note that all Hot Standby bugcheck dumps are written to the corresponding bugcheck dump file; bugcheck dumps are not written to the file you specify with the Output qualifier.

6.8.4 Incomplete Description of Protection Qualifier for RMU Backup After_Journal Command

The description of the Protection Qualifier for the RMU Backup After_Journal command is incomplete in the Oracle RMU Reference Manual for Digital UNIX. The complete description is as follows:

The Protection qualifier specifies the system file protection for the backup file produced by the RMU Backup After_Journal command. If you do not specify the Protection qualifier, the default access permissions are −rw−r−−−−− for backups to disk or tape.

Tapes do not allow delete or execute access and the superuser account always has both read and write access to tapes. In addition, a more restrictive class accumulates the access rights of the less restrictive classes.

If you specify the Protection qualifier explicitly, the differences in access permissions applied for backups to tape or disk as noted in the preceding paragraph are applied. Thus, if you specify Protection=(S,O,G:W,W:R), the access permissions on tape becomes rw−rw−r−.

6.8.5 RMU Extract Command Options Qualifier

A documentation error exists in the description of the Options=options−list qualifier of the RMU Extract command. Currently, the documentation states that this qualifier is not applied to output created by the Items=Volume qualifier. This is incorrect. Beginning with 6.1 of Oracle Rdb, the behavior of the Options=options−list qualifier is applied to output created by the Items=Volume qualifier.

6.8.6 RDM$SNAP_QUIET_POINT Logical is Incorrect

On page 2−72 of the Oracle RMU Reference Manual, the reference to the RDM$SNAP_QUIET_POINT logical is incorrect. The correct logical name is RDM$BIND_SNAP_QUIET_POINT.

6.8.7 Using Delta Time with RMU Show Statistics Command

Oracle RMU does not support the use of delta time. However, because the OpenVMS platform does, there is a workaround. You can specify delta time using the following syntax with the RMU Show Statistics command:

```
$ RMU/SHOW STATISTICS/OUTPUT=file-spec/UNTIL=" ' ' f$cvtime ("+7:00") ' "
```

The +7:00 adds 7 hours to the current time.

You can also use "TOMORROW" and "TODAY+n".

This information will be added to the description of the Until qualifier of the RMU Show Statistics command in a future release of the Oracle RMU Reference Manual.

6.9 Oracle Rdb7 Guide to Database Performance and Tuning

The following section provides corrected, clarified, or omitted information for the Oracle Rdb7 Guide to Database Performance and Tuning manual.

6.9.1 Dynamic OR Optimization Formats

In Table C−2 on Page C−7 of the Oracle Rdb7 Guide to Database Performance and Tuning, the dynamic OR optimization format is incorrectly documented as [l:h...]n. The correct formats for Oracle Rdb Release 7.0 and later are [(l:h)n] and [l:h,l2:h2].

6.9.2 Oracle Rdb Logical Names

The Oracle Rdb7 Guide to Database Performance and Tuning contains a table in Chapter 2 summarizing the Oracle Rdb logical names. The information in the following table supersedes the entries for the RDM$BIND_RUJ_ALLOC_BLKCNT and RDM$BIND_RUJ_EXTEND_BLKCNT logical names.

RDM$BIND_RUJ_ALLOC_BLKCNT Allows you to override the default value of the .ruj file. The block count value can be defined between 0 and 2 billion with a default of 127.

RDM$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 65535 with a default of 127.

6.9.3 Waiting for Client Lock Message

The Oracle Rdb7 Guide to Database Performance and Tuning contains a section in Chapter 3 that describes the Performance Monitor Stall Messages screen. The section contains a list describing the "Waiting for" messages. The description of the "waiting for client lock" message was missing from the list.

A client lock indicates that an Rdb metadata lock is in use. The term client indicates that Rdb is a client of the Rdb locking services. The metadata locks are used to guarantee memory copies of the metadata (table, index and column definitions) are consistent with the on−disk versions.

The "waiting for client lock" message means the database user is requesting an incompatible locking mode. For example, when trying to drop a table which is in use, the drop operation requests a PROTECTED WRITE lock on the metadata object (such as a table) which is incompatible with the existing PROTECTED READ lock currently used by other users of the table.

The lock name for these special locks consist of an encoded 16 byte name. The first 4 bytes contains the leading four bytes of the user name (for system objects the RDB$ prefix is skipped) followed by three longwords. The lock is displayed in text format first − here will be seen the prefix for the table, routine, or module name; followed by its hexadecimal representation. The text version masks out non−printable characters with a dot (.).

```
waiting for client '...."...EMPL' 4C504D450000002200000000400000055
```

The leftmost value seen in the hexadecimal output contains the name prefix which is easier read in the text field. Then comes a hex number (00000022) which is the id of the object. The id is described below for tables, views, functions, procedures, modules, and sequences.

- For tables and views, the id represents the unique value found in the RDB$RELATION_ID column of the RDB$RELATIONS system relation for the given table.
- For routines (that is functions and procedures), the id represents the unique value found in the RDB$ROUTINE_ID column of the RDB$ROUTINES system relation for the given routine.
- For modules, the id represents the unique value found in the RDB$MODULE_ID column of the RDB$MODULES system relation for the given module.
- For sequences, the id represents the unique value found in the RDB$SEQUENCE_ID column of the RDB$SEQUENCES system relation for the given sequence.

The next value displayed signifies the object type. The following table describes objects and their hexadecimal type values.

Table 6−2 Objects and Their Hexadecimal Type Value

Object	Hexadecimal Value
Tables or views	00000004
Modules	00000015
Routines	00000016
Sequences	00000019

The last value in the hexadecimal output represents the lock type. The hexadecimal value 55 indicates this is a client lock and distinct from page and other data structure locks.

The following example shows a "waiting for client lock" message from a Stall Messages screen while the application was processing the EMPLOYEES table from MF_PERSONNEL. The terminal should be set to 132 characters wide to view the full client lock string.

```
Process.ID  Since.................. T Stall.reason.............................Lock.ID.
27800643:1                            waiting for logical area 79 (CW)         16004833
27800507:1  31−OCT−2002 16:05:15.71 W waiting for client '...."...EMPL' 4C504D45000000220000000
```

To determine the name of the referenced object given the lock ID, the following queries can be used based on the object type:

```
SQL> select RDB$RELATION_NAME from RDB$RELATIONS where RDB$RELATION_ID = 25;
SQL> select RDB$MODULE_NAME from RDB$MODULES where RDB$MODULE_ID = 12;
SQL> select RDB$ROUTINE_NAME from RDB$ROUTINES where RDB$ROUTINE_ID = 7;
SQL> select RDB$SEQUENCE_NAME from RDB$SEQUENCES where RDB$SEQUENCE_ID = 2;
```

For more detailed lock information, perform the following steps:

- Press the L option from the horizontal menu to display a menu of lock IDs.
- Select the desired lock ID.

6.9.4 RDMS$TTB_HASH_SIZE Logical Name

The logical name RDMS$TTB_HASH_SIZE sets the size of the hash table used for temporary tables. If the logical name is not defined, Oracle Rdb uses a default value of 1249.

If you expect that temporary tables will be large (that is, 10K or more rows), use this logical name to adjust the hash table size to avoid long hash chains. Set the value to approximately 1/4 of the expected maximum number of rows for each temporary table. For example, if a temporary table will be populated with 100,000 rows, define this logical name to be 25000. If there are memory constraints on your system, you should define the logical name to be no higher than this value (1/4 of the expected maximum number of rows).

6.9.5 Error in Updating and Retrieving a Row by Dbkey Example 3−22

Example 3−22 in Section 3.8.3 that shows how to update and retrieve a row by dbkey is incorrect. The example should appear as follows:

```
SQL> ATTACH 'FILENAME MF_PERSONNEL.RDB';
SQL> --
SQL> -- Declare host variables
SQL> --
SQL> DECLARE :hv_row INTEGER;              -- Row counter
SQL> DECLARE :hv_employee_id ID_DOM;       -- EMPLOYEE_ID field
SQL> DECLARE :hv_employee_id_ind SMALLINT; -- Null indicator variable
SQL> --
SQL> DECLARE :hv_dbkey CHAR(8);            -- DBKEY storage
SQL> DECLARE :hv_dbkey_ind SMALLINT;       -- Null indicator variable
SQL> --
SQL> DECLARE :hv_last_name LAST_NAME_DOM;
SQL> DECLARE :hv_new_address_data_1 ADDRESS_DATA_1_DOM;
SQL> --
SQL> SET TRANSACTION READ WRITE;
SQL> BEGIN
cont> --
cont> -- Set the search value for SELECT
cont> --
cont>  SET :hv_last_name = 'Ames';
cont> --
cont> -- Set the NEW_ADDRESS_DATA_1 value
cont> --
cont> SET :hv_new_address_data_1 = '100 Broadway Ave.';
cont> END;
SQL> COMMIT;
SQL> --
SQL> SET TRANSACTION READ ONLY;
SQL> BEGIN
cont> SELECT E.EMPLOYEE_ID, E.DBKEY
cont>   INTO :hv_employee_id INDICATOR :hv_employee_id_ind,
cont>        :hv_dbkey INDICATOR :hv_dbkey_ind
cont>   FROM EMPLOYEES E
cont> WHERE E.LAST_NAME = :hv_last_name
cont> LIMIT TO 1 ROW;
cont> --
cont> GET DIAGNOSTICS :hv_row = ROW_COUNT;
cont> END;
SQL> COMMIT;
SQL> --
SQL> SET TRANSACTION READ WRITE RESERVING EMPLOYEES FOR SHARED WRITE;
SQL> BEGIN
cont> IF (:hv_row = 1) THEN
cont>    BEGIN
cont>    UPDATE EMPLOYEES E
cont>      SET E.ADDRESS_DATA_1 = :hv_new_address_data_1
cont>    WHERE E.DBKEY = :hv_dbkey;
cont>    END;
cont> END IF;
cont> END;
SQL> COMMIT;
SQL> --
SQL> -- Display result of change
SQL> --
SQL> SET TRANSACTION READ ONLY;
SQL> SELECT E.*
cont> FROM EMPLOYEES E
cont> WHERE E.DBKEY = :hv_dbkey;
 EMPLOYEE_ID   LAST_NAME        FIRST_NAME   MIDDLE_INITIAL
   ADDRESS_DATA_1              ADDRESS_DATA_2         CITY
      STATE   POSTAL_CODE   SEX    BIRTHDAY      STATUS_CODE
 00416         Ames             Louie        A
   100 Broadway Ave.                                  Alton
      NH      03809         M      13-Apr-1941   1

1 row selected
SQL>
```

The new example will appear in a future publication of the Oracle Rdb7 Guide to Database Performance and Tuning manual.

6.9.6 Error in Calculation of Sorted Index in Example 3−46

Example 3−46 in Section 3.9.5.1 shows the output when you use the RMU Analyze Indexes command and specify the Option=Debug qualifier and the DEPARTMENTS_INDEX sorted index.

The description of the example did not include the 8 byte dbkey in the calculation of the sorted index. The complete description is as follows:

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 422 bytes to 188 bytes and the percent fullness of the node records has dropped from 98 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 sort key is 4 bytes plus a null byte for a total of 5 bytes. The prefix is 1 byte and the suffix is 1 byte. The prefix indicates the number of bytes in the preceding key that are the same and the suffix indicates the number of bytes that are different from the preceding key. The dbkey pointer to the row is 8 bytes. There are 26 data rows multiplied by 15 bytes for a total of 390 bytes. The 15 bytes include:

- 7 bytes for the sort key: length + null byte + prefix + suffix
- 8 bytes for the dbkey pointer to the row

Add 32 bytes for index header and trailer information for the index node to the 390 bytes for a total of 422 bytes used. Index compression reduces the number of bytes used to 188 bytes used.

The revised description will appear in a future publication of the Oracle Rdb7 Guide to Database Performance and Tuning manual.

6.9.7 Documentation Error in Section C.7

The Oracle Rdb Guide to Database Performance And Tuning, Volume 2 contains an error in Section C.7 titled Displaying Sort Statistics with the R Flag.

When describing the output from this debugging flag, bullet 9 states:

- Work File Alloc indicates how many work files were used in the sort operation. A zero (0) value indicates that the sort was accomplished completely in memory.

This is incorrect, the statistics should be described as show below:

- Work File Alloc indicates how much space (in blocks) was allocated in the work files for this sort operation. A zero (0) value indicates that the sort was accomplished completely in memory.

This error will be corrected in a future release of Oracle Rdb Guide to Database Performance And Tuning.

6.9.8 Missing Tables Descriptions for the RDBEXPERT Collection Class

Appendix B in the Oracle Rdb7 Guide to Database Performance and Tuning describes the event−based data tables in the formatted database for the Oracle Rdb PERFORMANCE and RDBEXPERT collection classes. This section describes the missing tables for the RDBEXPERT collection class.

Table 6−3 shows the TRANS_TPB table.

Table 6−3 Columns for Table EPC$1_221_TRANS_TPB

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	
CLIENT_PC	INTEGER	
STREAM_ID	INTEGER	
TRANS_ID	VARCHAR(16)	
TRANS_ID_STR_ID	INTEGER	STR_ID_DOMAIN
TPB	VARCHAR(127)	
TPB_STR_ID	INTEGER	STR_ID_DOMAIN

Table 6−4 shows the TRANS_TPB_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 6−4 Columns for Table EPC$1_221_TRANS_TPB_ST

Column Name	Data Type	Domain
STR_ID	INTEGER	STR_ID_DOMAIN
SEGMENT_NUMBER	SMALLINT	SEGMENT_NUMBER_DOMAIN
STR_SEGMENT	VARCHAR(128)	

6.9.9 Missing Columns Descriptions for Tables in the Formatted Database

Some of the columns were missing from the tables in Appendix B in the Oracle Rdb7 Guide to Database Performance and Tuning. The complete table definitions are described in this section.

Table 6−5 shows the DATABASE table.

Table 6−5 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	
CLIENT_PC	INTEGER	
STREAM_ID	INTEGER	
DB_NAME	VARCHAR(255)	
DB_NAME_STR_ID	INTEGER	STR_ID_DOMAIN
IMAGE_FILE_NAME	VARCHAR(255)	
IMAGE_FILE_NAME_STR_ID	INTEGER	STR_ID_DOMAIN

Table 6−6 shows the REQUEST_ACTUAL table.

Table 6−6 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
PROM_DEADLOCKS_START	INTEGER
LOCK_RELS_START	INTEGER
LOCK_STALL_TIME_START	INTEGER
D_FETCH_RET_START	INTEGER
D_FETCH_UPD_START	INTEGER
D_LB_ALLOK_START	INTEGER
D_LB_GBNEEDLOCK_START	INTEGER
D_LB_NEEDLOCK_START	INTEGER
D_LB_OLDVER_START	INTEGER
D_GB_NEEDLOCK_START	INTEGER
D_GB_OLDVER_START	INTEGER
D_NOTFOUND_IO_START	INTEGER
D_NOTFOUND_SYN_START	INTEGER
S_FETCH_RET_START	INTEGER
S_FETCH_UPD_START	INTEGER
S_LB_ALLOK_START	INTEGER
S_LB_GBNEEDLOCK_START	INTEGER
S_LB_NEEDLOCK_START	INTEGER
S_LB_OLDVER_START	INTEGER
S_GB_NEEDLOCK_START	INTEGER
S_GB_OLDVER_START	INTEGER
S_NOTFOUND_IO_START	INTEGER
S_NOTFOUND_SYN_START	INTEGER
D_ASYNC_FETCH_START	INTEGER
S_ASYNC_FETCH_START	INTEGER
D_ASYNC_READIO_START	INTEGER
S_ASYNC_READIO_START	INTEGER

AS_READ_STALL_START	INTEGER	
AS_BATCH_WRITE_START	INTEGER	
AS_WRITE_STALL_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	
CLIENT_PC_END	INTEGER	
STREAM_ID_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	
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	
D_FETCH_RET_END	INTEGER	
D_FETCH_UPD_END	INTEGER	
D_LB_ALLOK_END	INTEGER	
D_LB_GBNEEDLOCK_END	INTEGER	
D_LB_NEEDLOCK_END	INTEGER	

D_LB_OLDVER_END	INTEGER
D_GB_NEEDLOCK_END	INTEGER
D_GB_OLDVER_END	INTEGER
D_NOTFOUND_IO_END	INTEGER
D_NOTFOUND_SYN_END	INTEGER
S_FETCH_RET_END	INTEGER
S_FETCH_UPD_END	INTEGER
S_LB_ALLOK_END	INTEGER
S_LB_GBNEEDLOCK_END	INTEGER
S_LB_NEEDLOCK_END	INTEGER
S_LB_OLDVER_END	INTEGER
S_GB_NEEDLOCK_END	INTEGER
S_GB_OLDVER_END	INTEGER
S_NOTFOUND_IO_END	INTEGER
S_NOTFOUND_SYN_END	INTEGER
D_ASYNC_FETCH_END	INTEGER
S_ASYNC_FETCH_END	INTEGER
D_ASYNC_READIO_END	INTEGER
S_ASYNC_READIO_END	INTEGER
AS_READ_STALL_END	INTEGER
AS_BATCH_WRITE_END	INTEGER
AS_WRITE_STALL_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 6−7 shows the TRANSACTION table.

Table 6−7 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	
CLIENT_PC_START	INTEGER	
STREAM_ID_START	INTEGER	
LOCK_MODE_START	INTEGER	
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	
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	
D_FETCH_RET_START	INTEGER	
D_FETCH_UPD_START	INTEGER	
D_LB_ALLOK_START	INTEGER	
D_LB_GBNEEDLOCK_START	INTEGER	
D_LB_NEEDLOCK_START	INTEGER	
D_LB_OLDVER_START	INTEGER	
D_GB_NEEDLOCK_START	INTEGER	
D_GB_OLDVER_START	INTEGER	
D_NOTFOUND_IO_START	INTEGER	
D_NOTFOUND_SYN_START	INTEGER	
S_FETCH_RET_START	INTEGER	
S_FETCH_UPD_START	INTEGER	
S_LB_ALLOK_START	INTEGER	
S_LB_GBNEEDLOCK_START	INTEGER	
S_LB_NEEDLOCK_START	INTEGER	
S_LB_OLDVER_START	INTEGER	
S_GB_NEEDLOCK_START	INTEGER	

S_GB_OLDVER_START	INTEGER	
S_NOTFOUND_IO_START	INTEGER	
S_NOTFOUND_SYN_START	INTEGER	
D_ASYNC_FETCH_START	INTEGER	
S_ASYNC_FETCH_START	INTEGER	
D_ASYNC_READIO_START	INTEGER	
S_ASYNC_READIO_START	INTEGER	
AS_READ_STALL_START	INTEGER	
AS_BATCH_WRITE_START	INTEGER	
AS_WRITE_STALL_START	INTEGER	
AREA_ITEMS_START	VARCHAR(128)	
AREA_ITEMS_START_STR_ID	INTEGER	STR_ID_DOMAIN
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	
CROSS_FAC_2_START	INTEGER	
CROSS_FAC_3_START	INTEGER	
CROSS_FAC_7_START	INTEGER	
CROSS_FAC_14_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	
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	
D_FETCH_RET_END	INTEGER	
D_FETCH_UPD_END	INTEGER	
D_LB_ALLOK_END	INTEGER	
D_LB_GBNEEDLOCK_END	INTEGER	
D_LB_NEEDLOCK_END	INTEGER	
D_LB_OLDVER_END	INTEGER	
D_GB_NEEDLOCK_END	INTEGER	
D_GB_OLDVER_END	INTEGER	
D_NOTFOUND_IO_END	INTEGER	
D_NOTFOUND_SYN_END	INTEGER	
S_FETCH_RET_END	INTEGER	
S_FETCH_UPD_END	INTEGER	
S_LB_ALLOK_END	INTEGER	
S_LB_GBNEEDLOCK_END	INTEGER	
S_LB_NEEDLOCK_END	INTEGER	
S_LB_OLDVER_END	INTEGER	
S_GB_NEEDLOCK_END	INTEGER	
S_GB_OLDVER_END	INTEGER	
S_NOTFOUND_IO_END	INTEGER	
S_NOTFOUND_SYN_END	INTEGER	
D_ASYNC_FETCH_END	INTEGER	
S_ASYNC_FETCH_END	INTEGER	
D_ASYNC_READIO_END	INTEGER	
S_ASYNC_READIO_END	INTEGER	
AS_READ_STALL_END	INTEGER	
AS_BATCH_WRITE_END	INTEGER	
AS_WRITE_STALL_END	INTEGER	
AREA_ITEMS_END	VARCHAR(128)	
AREA_ITEMS_END_STR_ID	INTEGER	STR_ID_DOMAIN
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	
CROSS_FAC_2_END	INTEGER	
CROSS_FAC_3_END	INTEGER	

CROSS_FAC_7_END	INTEGER
CROSS_FAC_14_END	INTEGER

Table 6−8 shows the REQUEST_BLR table.

Table 6−8 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	
CLIENT_PC	INTEGER	
STREAM_ID	INTEGER	
REQ_ID	INTEGER	
TRANS_ID	VARCHAR(16)	
TRANS_ID_STR_ID	INTEGER	STR_ID_DOMAIN
REQUEST_NAME	VARCHAR(31)	
REQUEST_NAME_STR_ID	INTEGER	STR_ID_DOMAIN
REQUEST_TYPE	INTEGER	
BLR	VARCHAR(127)	
BLR_STR_ID	INTEGER	STR_ID_DOMAIN

6.9.10 A Way to Find the Transaction Type of a Particular Transaction Within the Trace Database

The table EPC$1_221_TRANSACTION in the formatted Oracle Trace database has a column LOCK_MODE_START of longword datatype. The values of this column indicate the type of transaction a particular transaction was.

```
Value           Transaction type
-----           ----------------
8               Read only
9               Read write
14              Batch update
```

6.9.11 Using Oracle TRACE Collected Data

The following example shows how the OPTIMIZE AS clause is reflected in the Oracle TRACE database. When a trace collection is started the following SQL commands will record the request names.

```
SQL> attach `file personnel';
SQL> select last_name, first_name
cont> from employees
cont> optimize as request_one;
.
.
.
SQL> select employee_id
cont> from employees
cont> optimize as request_two;
.
.
.
SQL> select employee_id, city, state
cont> from employees
cont> optimize as request_three;
.
.
.
SQL> select last_name, first_name, employee_id, city, state
cont> from employees
cont> optimize as request_four;
.
.
.
```

Once an Oracle TRACE database has been populated from the collection, a query such as the following can be used to display the request names and types. The type values are described in Table 3−10. The unnamed queries in this example correspond to the queries executed by interactive SQL to validate the names of the tables an columns referenced in the user supplied queries.

```
SQL> select REQUEST_NAME, REQUEST_TYPE, TIMESTAMP_POINT
cont> from EPC$1_221_REQUEST_BLR;
REQUEST_NAME                      REQUEST_TYPE   TIMESTAMP_POINT
                                             1   15−JAN−1997 13:23:27.18
                                             1   15−JAN−1997 13:23:27.77
REQUEST_ONE                                  1   15−JAN−1997 13:23:28.21
REQUEST_TWO                                  1   15−JAN−1997 13:23:56.55
REQUEST_THREE                                1   15−JAN−1997 13:24:57.27
REQUEST_FOUR                                 1   15−JAN−1997 13:25:25.44
6 rows selected
```

The next example shows the internal query format (BLR) converted to SQL strings after EPC$EXAMPLES:EPC_BLR_TOSQL_CONVERTER.COM has been run.

```
SQL> SELECT A.REQUEST_NAME, B.SQL_STRING FROM
cont> EPC$1_221_REQUEST_BLR A,
cont> EPC$SQL_QUERIES B
cont> WHERE A.CLIENT_PC = 0 AND A.SQL_ID = B.SQL_ID;
A.REQUEST_NAME
  B.SQL_STRING
REQUEST_ONE
      SELECT C1.LAST_NAME, C1.FIRST_NAME.        FROM EMPLOYEES C1
. . .
REQUEST_TWO
      SELECT C1.EMPLOYEE_ID.        FROM EMPLOYEES C1
. . .
REQUEST_THREE
SELECT C1.EMPLOYEE_ID, C1.CITY, C1.STATE.        FROM EMPLOYEES C1
  .
  .
  .
```

```
4 rows selected
```

Table 4−17 shows the Request Types.

Table 6−9 Request Types

Symbolic Name	Value	Comment
RDB_K_REQTYPE_OTHER	0	A query executed internally by Oracle Rdb
RDB_K_REQTYPE_USER_REQUEST	1	A non−stored SQL statement, which includes compound statements
RDB_K_REQTYPE_PROCEDURE	2	A stored procedure
RDB_K_REQTYPE_FUNCTION	3	A stored function
RDB_K_REQTYPE_TRIGGER	4	A trigger action
RDB_K_REQTYPE_CONSTRAINT	5	A table or column constraint

6.9.12 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%.

In addition, you can determine how full a page must be before a duplicate node of size 112 will no longer fit. In this example, a database page can have up to 982−112−8 = 862 bytes in use before the page is too full. Therefore, if 862 or fewer bytes are in use, then enough space remains to store another small duplicates node. The threshold is then 862 / 982 = .8778, or 88%.

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, 55, 88));
```

These settings mean that any page at over 55% full will not be fetched when inserting a full index node, however, it may be fetched when inserting the smaller duplicates node. When the page is over 88% full then neither a full node nor a duplicate node can be stored, so the page is set as FULL. The lowest setting is not used and so can be set to any value less than or equal to the lowest used threshold.

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.

6.9.13 RDM$BIND_MAX_DBR_COUNT Documentation Clarification

Appendix A in Oracle Rdb7 Guide to Database Performance and Tuning incorrectly describes the use of the RDM$BIND_MAX_DBR_COUNT logical name.

Following is an updated description. Note that the difference in actual behavior between what is in the existing documentation and the software is that the logical name only controls the number of database recovery processes created at once during "node failure" recovery (that is, after a system or monitor crash or other abnormal shutdown).

When an entire database is abnormally shut down (due, for example, to a system failure), the database will have to be recovered in a "node failure" recovery mode. This recovery will be performed by another monitor in the cluster if the database is opened on another node or will be performed the next time the database is opened.

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 during a "node failure" recovery.

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 during a "node failure" recovery.

In a node failure recovery situation with the Row Cache feature enabled (regardless of the global buffer state), the database monitor will start a single database recovery (DBR) process to recover the Row Cache Server (RCS) process and all user processes from the oldest active checkpoint in the database.

6.10 Oracle Rdb7 Guide to SQL Programming

This section provides information that is missing or changed in the Oracle Rdb7 Guide to SQL Programming.

6.10.1 Location of Host Source File Generated by the SQL Precompiler

When the SQL precompiler generates host source files (for example, .c, .pas, or .for) from the precompiler source files, it locates these files based on the Object qualifier in the command given to the SQL precompiler.

The following examples show the location where the host source file is generated.

When the Object qualifier is not specified on the command line, the object and the host source file take the name of the SQL precompiler with the extensions of .obj and .c, respectively. For example:

```
$ sqlpre/cc scc_try_mli_successful.sc
$ dir scc_try_mli_successful.*

Directory MYDISK:[LUND]

SCC_TRY_MLI_SUCCESSFUL.C;1              SCC_TRY_MLI_SUCCESSFUL.OBJ;2
SCC_TRY_MLI_SUCCESSFUL.SC;2

Total of 3 files.
```

When the Object qualifier is specified on the command line, the object and the host source take the name given on the qualifier switch. It uses the default of the SQL precompiler source if a filespec is not specified. It uses the defaults of .obj and .c if the extension is not specified. If the host language is a language other than C, it uses the appropriate host source extension (for example, .pas or .for). The files also default to the current directory if a directory specification is not specified. For example:

```
$ sqlpre/cc/obj=myobj scc_try_mli_successful.sc
$ dir scc_try_mli_successful.*

Directory MYDISK:[LUND]

SCC_TRY_MLI_SUCCESSFUL.SC;2

Total of 1 file.
$ dir myobj.*

Directory MYDISK:[LUND]

MYOBJ.C;1           MYOBJ.OBJ;2

Total of 2 files.

$ sqlpre/cc/obj=MYDISK:[lund.tmp] scc_try_mli_successful.sc
$ dir scc_try_mli_successful.*

Directory MYDISK:[LUND]

SCC_TRY_MLI_SUCCESSFUL.SC;2

Total of 1 file.
```

```
$ dir MYDISK:[lund.tmp]scc_try_mli_successful.*

Directory MYDISK:[LUND.TMP]

SCC_TRY_MLI_SUCCESSFUL.C;1              SCC_TRY_MLI_SUCCESSFUL.OBJ;2

Total of 2 files.
```

6.10.2 Remote User Authentication

In the Oracle Rdb7 Guide to SQL Programming, Table 15−1 indicates that implicit authorization works from an OpenVMS platform to another OpenVMS platform using TCP/IP. This table is incorrect. Implicit authorization only works using DECnet in this situation.

The Oracle Rdb7 Guide to SQL Programming will be fixed in a future release.

6.10.3 Additional Information About Detached Processes

Oracle Rdb documentation omits necessary detail on running Oracle Rdb from a detached process.

Applications run from detached processes must ensure that the OpenVMS environment is established correctly before running Oracle Rdb, otherwise Oracle Rdb will not execute.

Attempts to attach to a database and execute an Oracle Rdb query from applications running as detached processes will result in an error similar to the following:

```
%RDB−F−SYS_REQUEST, error from system services request
−SORT−E−OPENOUT, error opening [file] as output
−RMS−F−DEV, error in device name or inappropriate device type for operation
```

The problem occurs because a detached process does not normally have the logical names SYS$LOGIN or SYS$SCRATCH defined.

There are two methods that can be used to correct this:

- Solution 1:
  Use the DCL command procedure RUN_PROCEDURE to run the ACCOUNTS application:
  RUN_PROCEDURE.COM includes the single line:
  $ RUN ACCOUNTS_REPORT
  Then execute this procedure using this command:
  $ RUN/DETACH/AUTHORIZE SYS$SYSTEM:LOGINOUT/INPUT=RUN_PROCEDURE
  This solution executes SYS$SYSTEM:LOGINOUT so that the command language interface (DCL) is activated. This causes the logical names SYS$LOGIN and SYS$SCRATCH to be defined for the detached process. The /AUTHORIZE qualifier also ensures that the users' process quota limits (PQLs) are used from the system authorization file rather than relying on the default PQL system parameters, which are often insufficient to run Oracle Rdb.
- Solution 2:
  If DCL is not desired, and SYS$LOGIN and SYS$SCRATCH are not defined, then prior to executing any Oracle Rdb statement, you should define the following logical names:
    - ♦ RDMS$BIND_WORK_FILE

Define this logical name to allow you to reduce the overhead of disk I/O operations for matching operations when used in conjunction with the RDMS$BIND_WORK_VM logical name. If the virtual memory file is too small then overflow to disk will occur at the disk and directory location specified by RDMS$BIND_WORK_FILE.
For more information on RDMS$BIND_WORK_FILE and RDMS$BIND_WORK_VM, see the Oracle Rdb Guide to Database Performance and Tuning.

♦ SORTWORK0, SORTWORK1, and so on
The OpenVMS Sort/Merge utility (SORT/MERGE) attempts to create sort work files in SYS$SCRATCH. If the SORTWORK logical names exist, the utility will not require the SYS$SCRATCH logical. However, note that not all queries will require sorting, and that some sorts will be completed in memory and so will not necessarily require disk space.
If you use the logical RDMS$BIND_SORT_WORKFILES, you will need to define further SORTWORK logical names as described in the Oracle Rdb Guide to Database Performance and Tuning.
You should also verify that sufficient process quotas are specified on the RUN/DETACH command line, or defined as system PQL parameters to allow Oracle Rdb to execute.

6.11 Guide to Using Oracle SQL/Services Client APIs

The following information describes Oracle SQL/Services documentation errors or omissions.

- The Guide to Using Oracle SQL/Services Client APIs does not describe changes to size and format of integer and floating−point data types
  Beginning with Oracle SQL/Services V5.1, the size and format of some integer and floating−point data types is changed as follows:
  - ♦ Trailing zeros occur in fixed−point numeric data types with SCALE FACTOR.
    Trailing zeros are now included after the decimal point up to the number of digits specified by the SCALE FACTOR. In versions of Oracle SQL/Services previous to V5.1, at most one trailing zero was included where the value was a whole number.
    The following examples illustrate the changes using a field defined as INTEGER(3):

```
V5.1 and     Versions previous
higher       to V5.1
--------     -----------------
  1.000        1.0
 23.400       23.4
567.890      567.89
```

  - ♦ Trailing zeros occur in floating−point data types. Trailing zeros are now included in the fraction, and leading zeros are included in the exponent, up to the maximum precision available, for fields assigned the REAL and DOUBLE PRECISION data types.

```
                                            Versions previous
Data Type           V5.1 and higher         to V5.1
----------------    ----------------------  -----------------
REAL                1.2340000E+01           1.234E+1
DOUBLE PRECISION    5.678900000000000E+001  5.6789E+1
```

  - ♦ Size of TINYINT and REAL data types is changed.
    The maximum size of the TINYINT and REAL data types is changed to correctly reflect the precision of the respective data types.
    The following table shows the maximum lengths of the data types now and in previous versions:

```
                    V5.1 and    Versions previous
Data type           higher      to V5.1
----------          --------    -----------------
TINYINT              4           6
REAL                15          24
```

- The Guide to Using Oracle SQL/Services Client APIs does not describe that the sqlsrv_associate() service returns SQL error code −1028 when connecting to a database service if the user has not been granted the right to attach to the database.
  When a user connects to a database service, the sqlsrv_associate() service completes with the SQL error code −1028, SQL_NO_PRIV, if the user has been granted access to the Oracle SQL/Services service, but has not been granted the right to attach to the database. A record of the failure is written to the executor process's log file. Note that the sqlsrv_associate() service completes with the Oracle SQL/Services error code −2034, SQLSRV_GETACCINF if the user has not been granted access to the Oracle SQL/Services service.

Chapter 7
Known Problems and Restrictions

This chapter describes problems and restrictions relating to Oracle Rdb Release 7.1.4, and includes workarounds where appropriate.

7.1 Known Problems and Restrictions in All Interfaces

This section describes known problems and restrictions that affect all interfaces for Release 7.1.

7.1.1 RDO IMPORT Does Not Support FORWARD_REFERENCES Created by SQL EXPORT

Recent versions of SQL EXPORT have included support for new features as they are added to Oracle Rdb. In particular, SQL now generates forward references for routines to allow references in the metadata prior to those routines being created. No such enhancements will be made to RDO IMPORT.

Due to changes in the CREATE STORAGE MAP statement for this release, the RDO IMPORT command is no longer able to process interchange (.RBR) files created by SQL EXPORT due to forward references to new storage mapping routines. See Section 4.1.5, Enhancements to CREATE and ALTER STORAGE MAP for details.

Note

The RDO IMPORT command has been deprecated since the release of Oracle Rdb V7.0. Oracle recommends that users change all scripts to use the SQL IMPORT command in the future.

The following example shows the reported error:

```
$ RDO
IMPORT 'thresh_alter_sql' INTO 'thresh' DICTIONARY IS NOT USED.
%RDO−W−UNSIMPORT, RDO IMPORT does not support all Oracle Rdb features, please
use SQL IMPORT
Exported by Oracle Rdb V7.1−301 Import/Export utility
A component of Oracle Rdb SQL V7.1−301
   .
   .
   .
IMPORTing STORAGE AREA: RDB$SYSTEM
IMPORTing STORAGE AREA: AREA1
IMPORTing STORAGE AREA: AREA2
IMPORTing STORAGE AREA: DEFAULT_AREA
%RDO−E−EXTRADATA, unexpected data at the end of the RBR file
```

With the current release, you can work around this problem in one of the following ways:

1. Change the command to execute the SQL IMPORT command. This is the recommended and long term solution to this problem.
2. Change the SQL EXPORT command to include the NO FORWARD_REFERENCES clause. This will eliminate the definitions which currently cause errors in RDO IMPORT. However, this interchange file may then not contain sufficient information to fully import the database.
3. The RMU/LOAD command can also be used to extract the data for individual tables. You must use the /MATCH_NAME qualifier for Load.

7.1.2 New Attributes Saved by RMU/LOAD Incompatible With Prior Versions

Bug 2676851

To improve the behavior of unloading views, Oracle Rdb Release 7.1.2 changed the way view columns were unloaded so that attributes for view computed columns, COMPUTED BY and AUTOMATIC columns were saved. These new attributes are not accepted by prior releases of Oracle Rdb.

The following example shows the reported error trying to load a file from V7.1.2 under V7.1.0.4.

```
%RMU−F−NOTUNLFIL, Input file was not created by RMU UNLOAD
%RMU−I−DATRECSTO,   0 data records stored.
%RMU−F−FTL_LOAD, Fatal error for LOAD operation at 21−OCT−2003 16:34:54.20
```

You can workaround this problem by using the /RECORD_DEFINITION qualifier and specifying the FORMAT=DELIMITED option. However, this technique does not support LIST OF BYTE VARYING column unloading.

7.1.3 RDMS−E−RTNSBC_INITERR, Cannot init. external routine server site executor

Execution of an external function or procudure with server site binding may unexpectedly fail.

The following example shows this problem.

```
%RDB−E−EXTFUN_FAIL, external routine failed to compile or execute successfully
−RDMS−E−EXTABORT, routine NNNNNNNNN execution has been aborted
−RDMS−E−RTNSBC_INITERR, Cannot init. external routine server site executor;
reason XX
```

In this example, NNNNNNNNN is the function name and XX is a decimal value such as 41.

While such errors are possible they are very unlikely to be seen, especially on systems that have had Rdb successfully installed. These errors usually indicate a problem with the environment. For instance, ensure that images RDMXSMvv.EXE, RDMXSRvv.EXE and RDMXSMPvv.EXE (where vv is the Rdb version) are installed and have the correct protections, as in the following example.

```
Directory DISK$:<SYS6.SYSCOMMON.SYSLIB>

RDMXSM70.EXE;3           183   8−APR−2004 09:37:31.36  (RWED,RWED,RWED,RE)
RDMXSMP70.EXE;3          159   8−APR−2004 09:37:31.54  (RWED,RWED,RWED,RE)
RDMXSR70.EXE;3            67   8−APR−2004 09:37:31.74  (RWED,RWED,RWED,RE)

Total of 3 files, 409 blocks.

DISK$:<SYS6.SYSCOMMON.SYSLIB>.EXE
   RDMXSM70;3       Open Hdr Shared            Lnkbl
DISK$:<SYS6.SYSCOMMON.SYSLIB>.EXE
   RDMXSMP70;3      Open Hdr Shared       Prot Lnkbl  Safe
DISK$:<SYS6.SYSCOMMON.SYSLIB>.EXE
   RDMXSR70;3       Open Hdr Shared            Lnkbl
```

7.1.4 SYSTEM−F−INSFMEM Fatal Error With SHARED MEMORY IS SYSTEM or LARGE MEMORY IS ENABLED in Galaxy Environment

When using the GALAXY SUPPORT IS ENABLED feature in an OpenVMS Galaxy environment, a *%SYSTEM−F−INSFMEM, insufficient dynamic memory error* may be returned when mapping record caches or opening the database. One source of this problem specific to a Galaxy configuration is running out of Galaxy Shared Memory regions. For Galaxy systems, GLX_SHM_REG is the number of shared memory region structures configured into the Galaxy Management Database (GMDB).

While the default value (for OpenVMS versions through at least V7.3−1) of 64 regions might be adequate for some installations, sites using a larger number of databases or row caches when the SHARED MEMORY IS SYSTEM or LARGE MEMORY IS ENABLED features are enabled may find the default insufficient.

If a *%SYSTEM−F−INSFMEM, insufficient dynamic memory* error is returned when mapping record caches or opening databases, Oracle Corporation recommends that you increase the GLX_SHM_REG parameter by 2 times the sum of the number of row caches and number of databases that might be accessed in the Galaxy at one time. As the Galaxy shared memory region structures are not very large, setting this parameter to a higher than required value does not consume a significant amount of physical memory. It also may avoid a later reboot of the Galaxy environment. This parameter must be set on all nodes in the Galaxy.

Galaxy Reboot Required

Changing the GLX_SHM_REG system parameter requires that the OpenVMS Galaxy environment be booted from scratch. That is, all nodes in the Galaxy must be shut down and then the Galaxy reformed by starting each instance.

7.1.5 Oracle Rdb and OpenVMS ODS−5 Volumes

The OpenVMS Version 7.2 release introduced an Extended File Specifications feature, which consists of two major components:

- A new, optional, volume structure, ODS−5, which provides support for file names that are longer and have a greater range of legal characters than in previous versions of OpenVMS.
- Support for "deep" directory trees.

ODS−5 was introduced primarily to provide enhanced file sharing capabilities for users of Advanced Server for OpenVMS 7.2 (formerly known as PATHWORKS for OpenVMS), as well as DCOM and JAVA applications.

In some cases, Oracle Rdb performs its own file and directory name parsing and explicitly requires ODS−2 (the traditional OpenVMS volume structure) file and directory name conventions to be followed. Because of this knowledge, Oracle does not support any Oracle Rdb database file components (including root files, storage area files, after image journal files, record cache backing store files, database backup files, after image journal backup files, etc.) that utilize any non−ODS−2 file naming features. For this reason, Oracle recommends that Oracle Rdb database components not be located on ODS−5 volumes.

Oracle does support Oracle Rdb database file components on ODS−5 volumes provided that all of these files and directories used by Oracle Rdb strictly follow the ODS−2 file and directory name conventions. In particular, all file names must be specified entirely in uppercase and "special" characters in file or directory names are forbidden.

7.1.6 Optimization of Check Constraints

Bug 1448422

When phrasing constraints using the "CHECK" syntax, a poorer strategy can be chosen by the optimizer than when the same or similar constraint is phrased using referential integrity (PRIMARY and FOREIGN KEY) constraints.

For example, I have two tables T1 and T2, both with one column, and I wish to ensure that all values in table T1 exist in T2. Both tables have an index on the referenced field. I could use a PRIMARY KEY constraint on T2 and a FOREIGN KEY constraint on T1.

```
SQL> alter table t2
cont>   alter column f2 primary key not deferrable;
SQL> alter table t1
cont>   alter column f1 references t2 not deferrable;
```

When deleting from the PRIMARY KEY table, Rdb will only check for rows in the FOREIGN KEY table where the FOREIGN KEY has the deleted value. This can be seen as an index lookup on T1 in the retrieval strategy.

```
SQL> delete from t2 where f2=1;
Get     Temporary relation      Retrieval by index of relation T2
  Index name  I2 [1:1]
Index only retrieval of relation T1
  Index name  I1 [1:1]
%RDB−E−INTEG_FAIL, violation of constraint T1_FOREIGN1 caused operation to fail
```

The failure of the constraint is not important. What is important is that Rdb efficiently detects that only those rows in T1 with the same values as the deleted row in T2 can be affected.

It is necessary sometimes to define this type of relationship using CHECK constraints. This could be necessary because the presence of NULL values in the table T2 precludes the definition of a primary key on that table. This could be done with a CHECK constraint of the form:

```
SQL> alter table t1
cont>   alter column f1
cont>   check (f1 in (select * from t2)) not deferrable;
SQL> delete from t2 where f2=1;
Get     Temporary relation      Retrieval by index of relation T2
  Index name  I2 [1:1]
Cross block of 2 entries
  Cross block entry 1
    Index only retrieval of relation T1
      Index name  I1 [0:0]
  Cross block entry 2
    Conjunct        Aggregate−F1    Conjunct
    Index only retrieval of relation T2
      Index name  I2 [0:0]
%RDB−E−INTEG_FAIL, violation of constraint T1_CHECK1 caused operation to fail
```

The cross block is for the constraint evaluation. This retrieval strategy indicates that to evaluate the constraint, the entire index on table T1 is being scanned and for each key, the entire index in table T2 is being scanned. The behavior can be improved somewhat by using an equality join condition in the select clause of the constraint:

```
SQL> alter table t1
cont>   alter column f1
cont>   check (f1 in (select * from t2 where f2=f1))
cont>      not deferrable;

or:

SQL> alter table t1
cont>   alter column f1
cont>   check (f1=(select * from t2 where f2=f1))
cont>      not deferrable;
```

In both cases the retrieval strategy will look like this:

```
SQL> delete from t2 where f2=1;
Get     Temporary relation      Retrieval by index of relation T2
  Index name  I2 [1:1]
Cross block of 2 entries
  Cross block entry 1
    Index only retrieval of relation T1
      Index name  I1 [0:0]
  Cross block entry 2
    Conjunct        Aggregate−F1    Conjunct
    Index only retrieval of relation T2
      Index name  I2 [1:1]
%RDB−E−INTEG_FAIL, violation of constraint T1_CHECK1 caused operation to fail
```

While the entire T1 index is scanned, at least the value from T1 is used to perform an index lookup on T2.

These restrictions result from semantic differences in the behavior of the "IN" and "EXISTS" operators with respect to null handling, and the complexity of dealing with non−equality join conditions.

To improve the performance of this type of integrity check on larger tables, it is possible to use a series of triggers to perform the constraint check. The following triggers perform a similar check to the constraints above.

```
SQL> create trigger t1_insert
cont>  after insert on t1
cont>  when (not exists (select * from t2 where f2=f1))
cont>    (error) for each row;
SQL> create trigger t1_update
cont>  after update on t1
cont>  when (not exists (select * from t2 where f2=f1))
cont>    (error) for each row;
SQL> ! A delete trigger is not needed on T1.
SQL> create trigger t2_delete
cont>  before delete on t2
cont>  when (exists (select * from t1 where f1=f2))
cont>    (error) for each row;
SQL> create trigger t2_modify
cont>  after update on t2
cont>  referencing old as t2o new as t2n
cont>  when (exists (select * from t1 where f1=t2o.f2))
cont>    (error) for each row;
SQL> ! An insert trigger is not needed on T2.
```

The strategy for a delete on T2 is now:

```
SQL> delete from t2 where f2=1;
Aggregate−F1    Index only retrieval of relation T1
  Index name  I1 [1:1]
Temporary relation      Get     Retrieval by index of relation T2
  Index name  I2 [1:1]
%RDB−E−TRIG_INV_UPD, invalid update; encountered error condition defined for
trigger
−RDMS−E−TRIG_ERROR, trigger T2_DELETE forced an error
```

The trigger strategy is the index only retrieval displayed first. You will note that the index on T1 is used to examine only those rows that may be affected by the delete.

Care must be taken when using this workaround as there are semantic differences in the operation of the triggers, the use of "IN" and "EXISTS", and the use of referential integrity constraints.

This workaround is useful where the form of the constraint is more complex, and cannot be phrased using referential integrity constraints. For example, if the application is such that the value in table T1 may be spaces or NULL to indicate the absence of a value, the above triggers could easily be modified to allow for these semantics.

7.1.7 Using Databases from Releases Earlier Than V6.0

You cannot convert or restore databases earlier than V6.0 directly to V7.1. The RMU Convert command for V7.1 supports conversions from V6.0 through V7.0 only. If you have a V3.0 through V5.1 database, you must convert it to at least V6.0 and then convert it to V7.1. For example, if you have a V4.2 database, convert it first to at least V6.0, then convert the resulting database to V7.1.

If you attempt to convert a database created prior to V6.0 directly to V7.1, Oracle RMU generates an error.

7.1.8 Carryover Locks and NOWAIT Transaction Clarification

In NOWAIT transactions, the BLAST (Blocking AST) mechanism cannot be used. For the blocking user to receive the BLAST signal, the requesting user must request the locked resource with WAIT (which a NOWAIT transaction does not do). Oracle Rdb defines a resource called NOWAIT, which is used to indicate that a NOWAIT transaction has been started. When a NOWAIT transaction starts, the user requests the NOWAIT resource. All other database users hold a lock on the NOWAIT resource so that when the NOWAIT transaction starts, all other users are notified with a NOWAIT BLAST. The BLAST causes blocking users to release any carryover locks. There can be a delay before the transactions with carryover locks detect the presence of the NOWAIT transaction and release their carryover locks. You can detect this condition by examining the stall messages. If the "Waiting for NOWAIT signal (CW)" stall message appears frequently, the application is probably experiencing a decrease in performance, and you should consider disabling the carryover lock behavior.

7.1.9 Unexpected Results Occur During Read−Only Transactions on a Hot Standby Database

When using Hot Standby, it is typical to use the standby database for reporting, simple queries, and other read−only transactions. If you are performing these types of read−only transactions on a standby database, be sure you can tolerate a READ COMMIT level of isolation. This is because the Hot Standby database might be updated by another transaction before the read−only transaction finishes, and the data retrieved might not be what you expected.

Because Hot Standby does not write to the snapshot files, the isolation level achieved on the standby database for any read−only transaction is a READ COMMITED transaction. This means that nonrepeatable reads and phantom reads are allowed during the read−only transaction:

- Nonrepeatable read operations: Allows the return of different results within a single transaction when an SQL operation reads the same row in a table twice. Nonrepeatable reads can occur when another transaction modifies and commits a change to the row between transactions. Because the standby database will update the data when it confirms a transaction has been committed, it is very possible to see an SQL operation on a standby database return different results.
- Phantom read operations: Allows the return of different results within a single transaction when an SQL operation retrieves a range of data values (or similar data existence check) twice. Phantoms can occur if another transaction inserted a new record and committed the insertion between executions of the range retrieval. Again, because the standby database may do this, phantom reads are possible.

Thus, you cannot rely on any data read from the standby database to remain unchanged. Be sure your read−only transactions can tolerate a READ COMMIT level of isolation before you implement procedures that read and use data from a standby database.

7.1.10 Both Application and Oracle Rdb Using SYS$HIBER

In application processes that use Oracle Rdb and the $HIBER system service (possibly through RTL routines such as LIB$WAIT), the application must ensure that the event being waited for has actually occurred. Oracle Rdb uses $HIBER/$WAKE sequences for interprocess communications particularly when the ALS (AIJ Log Server) feature is enabled.

The use of the $WAKE system service by Oracle Rdb can interfere with other users of $HIBER (such as the routine LIB$WAIT) that do not check for event completion, possibly causing a $HIBER to be unexpectedly resumed without waiting at all.

To avoid these situations, consider altering the application to use a code sequence that avoids continuing without a check for the operation (such as a delay or a timer firing) being complete.

The following pseudo−code shows how a flag can be used to indicate that a timed−wait has completed correctly. The wait does not complete until the timer has actually fired and set TIMER_FLAG to TRUE. This code relies on ASTs being enabled.

```
ROUTINE TIMER_WAIT:
    BEGIN
    ! Clear the timer flag
    TIMER_FLAG = FALSE
    ! Schedule an AST for sometime in the future
    STAT = SYS$SETIMR (TIMADR = DELTATIME, ASTRTN = TIMER_AST)
    IF STAT <> SS$_NORMAL
    THEN BEGIN
         LIB$SIGNAL (STAT)
         END
    ! Hibernate.  When the $HIBER completes, check to make
    ! sure that TIMER_FLAG is set indicating that the wait
    ! has finished.
    WHILE TIMER_FLAG = FALSE
    DO  BEGIN
        SYS$HIBER()
        END
    END
ROUTINE TIMER_AST:
    BEGIN
    ! Set the flag indicating that the timer has expired
    TIMER_FLAG = TRUE
    ! Wake the main-line code
    STAT = SYS$WAKE ()
    IF STAT <> SS$_NORMAL
    THEN BEGIN
         LIB$SIGNAL (STAT)
         END
    END
```

The LIB$K_NOWAKE flag can be specified when using the OpenVMS LIB$WAIT routine to allow an alternate wait scheme (using the $SYNCH system service) that can avoid potential problems with multiple code sequences using the $HIBER system service.

7.1.11 Bugcheck Dump Files with Exceptions at COSI_CHF_SIGNAL

In certain situations, Oracle Rdb bugcheck dump files indicate an exception at COSI_CHF_SIGNAL. This location is, however, not the address of the actual exception. The actual exception occurred at the previous call frame on the stack (the one listed as the next Saved PC after the exception).

For example, consider the following bugcheck file stack information:

```
$ SEARCH RDSBUGCHK.DMP "EXCEPTION","SAVED PC","-F-","-E-"

***** Exception at 00EFA828 : COSI_CHF_SIGNAL + 00000140
%COSI-F-BUGCHECK, internal consistency failure
Saved PC = 00C386F0 : PSIINDEX2JOINSCR + 00000318
Saved PC = 00C0BE6C : PSII2BALANCE + 0000105C
Saved PC = 00C0F4D4 : PSII2INSERTT + 000005CC
Saved PC = 00C10640 : PSII2INSERTTREE + 000001A0
      .
      .
      .
```

In this example, the exception actually occurred at PSIINDEX2JOINSCR offset 00000318. If you have a bugcheck dump with an exception at COSI_CHF_SIGNAL, it is important to note the next "Saved PC" because it is needed when working with Oracle Rdb Worldwide Support.

7.1.12 Read−only Transactions Fetch AIP Pages Too Often

Oracle Rdb read−only transactions fetch Area Inventory Pages (AIP) to ensure that the logical area has not been modified by an exclusive read−write transaction. This check is needed because an exclusive read−write transaction does not write snapshot pages and these pages may be needed by the read−only transaction.

Because AIPs are always stored in the RDB$SYSTEM area, reading the AIP pages could represent a significant amount of I/O to the RDB$SYSTEM area for some applications. Setting the RDB$SYSTEM area to read−only can avoid this problem, but it also prevents other online operations that might be required by the application so it is not a viable workaround in all cases.

This problem has been reduced in Oracle Rdb release 7.0. The AIP entries are now read once and then are not read again unless they need to be. This optimization requires that the carry−over locks feature be enabled (this is the default setting). If carry over locks are not enabled, this optimization is not enabled and the behavior is the same as in previous releases.

7.1.13 Row Cache Not Allowed While Hot Standby Replication is Active

The row cache feature may not be enabled on a hot standby database while replication is active. The hot standby feature will not start if row cache is enabled.

This restriction exists because rows in the row cache are accessed via logical dbkeys. However, information transferred to the standby database via the after image journal facility only contains physical dbkeys. Because there is no way to maintain rows in the cache via the hot standby processing, the row cache must be disabled when the standby database is open and replication is active.

A new command qualifier, ROW_CACHE=DISABLED, has been added to the RMU Open command. To open the hot standby database prior to starting replication, use the ROW_CACHE=DISABLED qualifier on the RMU Open command.

7.1.14 Excessive Process Page Faults and other Performance Considerations During Oracle Rdb Sorts

Excessive hard or soft page faulting can be a limiting factor of process performance. One factor contributing to Oracle Rdb process page faulting is sorting operations. Common causes of sorts include the SQL GROUP BY, ORDER BY, UNION, and DISTINCT clauses specified for a query, and index creation operations. Defining the logical name RDMS$DEBUG_FLAGS to "RS" can help determine when Oracle Rdb sort operations are occurring and to display the sort keys and statistics.

Oracle Rdb includes its own copy of the OpenVMS SORT32 code within the Oracle Rdb images and does not generally call the routines in the OpenVMS run−time library. A copy of the SORT32 code is used to provide stability between versions of Oracle Rdb and OpenVMS and because Oracle Rdb calls the sort routines from executive processor mode which is difficult to do using the SORT32 shareable image. SQL IMPORT and RMU Load operations do, however, call the OpenVMS SORT run−time library.

At the beginning of a sort operation, the SORT code allocates some memory for working space. The SORT code uses this space for buffers, in−memory copies of the data, and sorting trees.

SORT does not directly consider the processes quotas or parameters when allocating memory. The effects of WSQUOTA and WSEXTENT are indirect. At the beginning of each sort operation, the SORT code attempts to adjust the process working set to the maximum possible size using the $ADJWSL system service specifying a requested working set limit of %X7FFFFFFF pages (the maximum possible). SORT then uses a value of 75% of the returned working set for virtual memory scratch space. The scratch space is then initialized and the sort begins.

The initialization of the scratch space generally causes page faults to access the pages newly added to the working set. Pages that were in the working set already may be faulted out as the new pages are faulted in. Once the sort operation completes and SORT returns back to Oracle Rdb, the pages that may have been faulted out of the working set are likely to be faulted back into the working set.

When a process working set is limited by the working set quota (WSQUOTA) parameter and the working set extent (WSEXTENT) parameter is a much larger value, the first call to the sort routines can cause many page faults as the working set grows. Using a value of WSEXTENT that is closer to WSQUOTA can help reduce the impact of this case.

With some OpenVMS versions, AUTOGEN sets the SYSGEN parameter PQL_MWSEXTENT equal to the WSMAX parameter. This means that all processes on the system end up with WSEXTENT the same as WSMAX. Since that might be quite high, sorting might result in excessive page faulting. You may want to explicitly set PQL_MWSEXTENT to a lower value if this is the case on your system.

Sort work files are another factor to consider when tuning for Oracle Rdb sort operations. When the operation can not be done in the available memory, SORT uses temporary disk files to hold the data as it is being sorted. The Oracle Rdb7 Guide to Database Performance and Tuning contains more detailed information about sort work files.

The logical name RDMS$BIND_SORT_WORKFILES specifies how many work files sort is to use if work files are required. The default is 2 and the maximum number is 10. The work files can be individually controlled by the SORTWORKn logical names (where n is from 0 through 9). You can increase the efficiency of sort operations 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 your SYS$SCRATCH directory. By default, SYS$SCRATCH is the same device and directory as the SYS$LOGIN location. Spreading the I/O load over many disks improves efficiency as well as performance by taking advantage of the system resources and helps prevent disk I/O bottlenecks. Specifying that a your work files reside on separate disks permits overlap of the SORT read/write cycle. You may also encounter cases where insufficient space exists on the SYS$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.

Note that SORT uses the work files for different sorted runs, and then merges the sorted runs into larger groups. If the source data is mostly sorted, then not every sort work file may need to be accessed. This is a possible source of confusion because even with 10 sort work files, it is possible to exceed the capacity of the first SORT file and the sort operation fails never having accessed the remaining 9 sort work files.

Note that the logical names RDMS$BIND_WORK_VM and RDMS$BIND_WORK_FILE do not affect or control the operation of sort. These logical names are used to control other temporary space allocation within Oracle Rdb.

7.1.15 Control of Sort Work Memory Allocation

Oracle Rdb uses a built−in SORT32 package to perform many sort operations. Sometimes, these sorts exhibit a significant performance problem when initializing work memory to be used for the sort. This behavior can be experienced, for example, when a very large sort cardinality is estimated, but the actual sort cardinality is small.

In rare cases, it may be desirable to artificially limit the sort package's use of work memory. Two logicals have been created to allow this control. In general, there should be no need to use either of these logicals and misuse of them can significantly impact sort performance. Oracle recommends that these logicals be used carefully and sparingly.

The logical names are:

Table 7−1 Sort Memory Logicals

Logical	Definition
RDMS$BIND_SORT_MEMORY_WS_FACTOR	Specifies a percentage of the process's working set limit to be used when allocating sort memory for the built−in SORT32 package. If not defined, the default value is 75 (representing 75%), the maximum value is 75 (representing 75%), and the minimum value is 2 (representing 2%). Processes with vary large working set limits can sometimes experience significant page faulting and CPU consumption while initializing sort memory. This logical name can restrict the sort work memory to a percentage of the processes maximum working set.
RDMS$BIND_SORT_MEMORY_MAX_BYTES	Specifies an absolute limit to be used when allocating sort memory for the built−in SORT32 package. If not defined, the default value is unlimited (up to 1GB), the maximum value is 2,147,483,647 and the minimum value is 32,768.

7.1.16 The Halloween Problem

When a cursor is processing rows selected from a table, it is possible that another separate query can interfere with the retrieval of the cursor by modifying the index columns key values used by the cursor.

For instance, if a cursor selects all EMPLOYEES with LAST_NAME >= 'M', it is likely that the query will use the sorted index on LAST_NAME to retrieve the rows for the cursor. If an update occurs during the processing of the cursor which changes the LAST_NAME of an employee from "Mason" to "Rickard", then it is possible that that employee row will be processed twice. First when it is fetched with name "Mason", and then later when it is accessed by the new name "Rickard".

The Halloween problem is a well known problem in relational databases. Access strategies which optimize the

I/O requirements, such as Index Retrieval, can be subject to this problem. Interference from queries by other sessions are avoided by locking and are controlled by the ISOLATION LEVEL options in SQL, or the CONCURRENCY/CONSISTENCY options in RDO/RDML.

Oracle Rdb avoids this problem if it knows that the cursors subject table will be updated. For example, if the SQL syntax UPDATE ... WHERE CURRENT OF is used to perform updates of target rows, or the RDO/RDML MODIFY statement uses the context variable for the stream. Then the optimizer will choose an alternate access strategy if an update can occur which may cause the Halloween problem. This can be seen in the access strategy in Example 2−2 as a "Temporary relation" being created to hold the result of the cursor query.

When you use interactive or dynamic SQL, the UPDATE ... WHERE CURRENT OF or DELETE ... WHERE CURRENT OF statements will not be seen until after the cursor is declared and opened. In these environments, you must use the FOR UPDATE clause to specify that columns selected by the cursor will be updated during cursor processing. This is an indication to the Rdb optimizer so that it protects against the Halloween problem in this case. This is shown in Example 2−1 and Example 2−2.

The following example shows that the EMP_LAST_NAME index is used for retrieval. Any update performed will possibly be subject to the Halloween problem.

```
SQL> set flags 'strategy';
SQL> declare emp  cursor for
cont> select * from employees where last_name >= 'M'
cont> order by last_name;
SQL> open emp;
Conjunct        Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_LAST_NAME [1:0]
SQL> close emp;
```

The following example shows that the query specifies that the column LAST_NAME will be updated by some later query. Now the optimizer protects the EMP_LAST_NAME index used for retrieval by using a "Temporary Relation" to hold the query result set. Any update performed on LAST_NAME will now avoid the Halloween problem.

```
SQL> set flags 'strategy';
SQL> declare emp2 cursor for
cont> select * from employees where last_name >= 'M'
cont> order by last_name
cont> for update of last_name;
SQL> open emp2;
Temporary relation      Conjunct        Get
Retrieval by index of relation EMPLOYEES
  Index name  EMP_LAST_NAME [1:0]
SQL> close emp2;
```

When you use the SQL precompiler, or the SQL module language compiler it can be determined from usage that the cursor context will possibly be updated during the processing of the cursor because all cursor related statements are present within the module. This is also true for the RDML/RDBPRE precompilers when you use the DECLARE_STREAM and START_STREAM statements and use the same stream context to perform all MODIFY and ERASE statements.

The point to note here is that the protection takes place during the open of the SQL cursor (or RDO stream), not during the subsequent UPDATE or DELETE.

If you execute a separate UPDATE query which modifies rows being fetched from the cursor then the actual rows fetched will depend upon the access strategy chosen by the Rdb optimizer. As the query is separate from the cursors query (i.e. doesn't reference the cursor context), then the optimizer does not know that the cursor selected rows are potentially updated and so cannot perform the normal protection against the Halloween problem.

7.2 SQL Known Problems and Restrictions

This section describes known problems and restrictions for the SQL interface for release 7.1.

7.2.1 SET FLAGS CRONO_FLAG to be Removed

The SET FLAGS statement and RDMS$SET_FLAGS logical name currently accept the obsolete keyword CRONO_FLAG. This keyword will be removed in the next release of Oracle Rdb V7.1. Please update all scripts and applications to use the keyword CHRONO_FLAG.

7.2.2 Interchange File (RBR) Created by Oracle Rdb Release 7.1 Not Compatible With Previous Releases

To support the large number of new database attributes and objects, the protocol used by SQL EXPORT and SQL IMPORT has been enhanced to support more protocol types. Therefore, this format of the Oracle Rdb release 7.1 interchange files can no longer be read by older versions of Oracle Rdb.

Oracle Rdb continues to provide upward compatibility for interchange files generated by older versions.

Oracle Rdb has never supported backward compatibility, however, it was sometimes possible to use an interchange file with an older version of IMPORT. However, this protocol change will no longer permit this usage.

7.2.3 System Relation Change for International Database Users

Due to an error in creating the RDB$FIELD_VERSIONS system relation, another system relation, RDB$STORAGE_MAP_AREAS, cannot be accessed if the session character sets are not set to DEC_MCS.

This problem prevents the new Oracle Rdb GUIs, specifically the Oracle Rdb Schema Manager, from viewing indexes and storage maps from existing Oracle Rdb databases.

The problem can be easily corrected by executing the following SQL statement after attaching to the database:

```
SQL> UPDATE RDB$FIELD_VERSIONS SET RDB$FIELD_SUB_TYPE = 32767
cont> WHERE RDB$FIELD_NAME = 'RDB$AREA_NAME';
```

7.2.4 Single Statement LOCK TABLE is Not Supported for SQL Module Language and SQL Precompiler

The new LOCK TABLE statement is not currently supported as a single statement within the module language or embedded SQL language compiler.

Instead you must enclose the statement in a compound statement. That is, use BEGIN... END around the statement as shown in the following example. This format provides all the syntax and flexibility of LOCK TABLE.

This restriction does not apply to interactive or dynamic SQL.

The following extract from the module language listing file shows the reported error if you use LOCK TABLE as a single statement procedure. The other procedure in the same module is acceptable because it uses a compound statement that contains the LOCK TABLE statement.

```
1 MODULE sample_test
2 LANGUAGE C
3 PARAMETER COLONS
4
5 DECLARE ALIAS FILENAME 'mf_personnel'
6
7 PROCEDURE a (SQLCODE);
8 LOCK TABLE employees FOR EXCLUSIVE WRITE MODE;
%SQL−F−WISH_LIST, (1) Feature not yet implemented − LOCK TABLE requires compound
statement
9
10 PROCEDURE b (SQLCODE);
11 BEGIN
12 LOCK TABLE employees FOR EXCLUSIVE WRITE MODE;
13 END;
```

To workaround this problem of using LOCK TABLE for SQL module language or embedded SQL application, use a compound statement in an EXEC SQL statement.

7.2.5 Multistatement or Stored Procedures May Cause Hangs

Long−running multistatement or stored procedures can cause other users in the database to hang if the procedures obtain resources needed by those other users. Some resources obtained by the execution of a multistatement or stored procedure are not released until the multistatement or stored procedure finishes. Thus, any−long running multistatement or stored procedure can cause other processes to hang. This problem can be encountered even if the statement contains SQL COMMIT or ROLLBACK statements.

The following example demonstrates the problem. The first session enters an endless loop; the second session attempts to backup the database but hangs forever.

```
Session 1:

SQL> attach 'filename MF_PERSONNEL';
SQL> create function LIB$WAIT (in real by reference)
cont> returns integer;
cont> external name LIB$WAIT location 'SYS$SHARE:LIBRTL.EXE'
cont> language general general parameter style variant;
SQL> commit;
        .
        .
        .
$ SQL
SQL> attach 'filename MF_PERSONNEL';
SQL> begin
cont> declare :LAST_NAME LAST_NAME_DOM;
cont> declare :WAIT_STATUS integer;
cont> loop
cont> select LAST_NAME into :LAST_NAME
cont> from EMPLOYEES where EMPLOYEE_ID = '00164';
cont> rollback;
cont> set :WAIT_STATUS = LIBWAIT (5.0);
cont> set transaction read only;
cont> end loop;
cont> end;

Session 2:

$ RMU/BACKUP/LOG/ONLINE MF_PERSONNEL MF_PERSONNEL

From a third session, you can see that the backup process is waiting
for a lock held in the first session:

$ RMU/SHOW LOCKS /MODE=BLOCKING MF_PERSONNEL
    .
    .
    .
Resource: nowait signal

ProcessID Process Name        Lock ID  System ID Requested Granted
--------- ---------------     --------- -------- --------- -------
20204383  RMU BACKUP.....     5600A476  00010001 CW        NL
2020437B  SQL............     3B00A35C  00010001 PR        PR
```

There is no workaround for this restriction. When the multistatement or stored procedure finishes execution, the resources needed by other processes are released.

7.2.6 Use of Oracle Rdb from Shareable Images

If code in the image initialization routine of a shareable image makes any calls into Oracle Rdb, through SQL or any other means, access violations or other unexpected behavior may occur if Oracle Rdb images have not had a chance to do their own initialization.

To avoid this problem, applications must take one of the following steps:

- Do not make Oracle Rdb calls from the initialization routines of shareable images.
- Link in such a way that the RDBSHR.EXE image initializes first. You can do this by placing the reference to RDBSHR.EXE and any other Oracle Rdb shareable images last in the linker options file.

This is not a bug; it is a restriction resulting from the way OpenVMS image activation works.

7.3 Oracle RMU Known Problems and Restrictions

This section describes known problems and restrictions for the RMU interface for release 7.1.

7.3.1 RMU/BACKUP MAX_FILE_SIZE Option Has Been Disabled

The MAX_FILE_SIZE option of the RMU/BACKUP/DISK_FILE qualifier for backup to multiple disk files has been temporarily disabled since it creates corrupt RBF files if the maximum file size in megabytes is exceeded and a new RBF file is created. It also does not give a unique name to the new RBF file but creates an RBF file with the same name but a new version number in the same disk directory. This will cause an RMU−F−BACFILCOR error on the restore and the restore will not complete.

The multi−file disk backup and restore will succeed if this option is not used. If this option is specified, a warning message is now output that this qualifier will be ignored.

The following example shows that the MAX_FILE_SIZE option, when used with the /DISK_FILE qualifier on an RMU/BACKUP, will be ignored and a warning message will be output.

```
$ RMU/BACKUP /ONLINE                    -
             /NOCRC                     -
             /NOLOG                     -
             /NOINCREMENTAL             -
             /QUIET_POINT               -
             TEST_DB_DIR:TEST_DB
  -
BACKUP_DIR_1:TEST_DB/DISK_FILE=(WRITER_THREADS=3,MAX_FILE_SIZE=10) ,-
BACKUP_DIR_2:/DISK_FILE=(WRITER_THREADS=3,MAX_FILE_SIZE=10) ,-
BACKUP_DIR_3:/DISK_FILE=(WRITER_THREADS=3,MAX_FILE_SIZE=10)

%RMU−W−DISABLEDOPTION, The MAX_FILE_SIZE option is temporarily disabled
        and will be ignored
```

As a workaround to avoid this problem, do not specify the MAX_FILE_SIZE option with the /DISK_FILE qualifier.

7.3.2 RMU Convert Fails When Maximum Relation ID is Exceeded

If, when relation IDs are assigned to new system tables during an RMU Convert of an Oracle Rdb V7.0 database to a V7.1 database, the maximum relation ID of 8192 allowed by Oracle Rdb is exceeded, the fatal error %RMU−F−RELMAXIDBAD is displayed and the database is rolled back to V70. Contact your Oracle support representative if you get this error. Note that when the database is rolled back, the fatal error %RMU−F−CVTROLSUC is displayed to indicate that the rollback was successful but caused by the detection of a fatal error and not requested by the user.

This condition only occurs if there are an extremely large number of tables defined in the database or if a large number of tables were defined but have subsequently been deleted.

The following example shows both the %RMU−F−RELMAXIDBAD error message if the allowed database relation ID maximum of 8192 is exceeded and the %RMU−F−CVTROLSUC error message when the database has been rolled back to V7.0 since it cannot be converted to V7.1:

```
$rmu/convert mf_personnel
%RMU−I−RMUTXT_000, Executing RMU for Oracle Rdb V7.1−00
Are you satisfied with your backup of
 DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1 and your backup of
 any associated .aij files [N]? Y
 %RMU−I−LOGCONVRT, database root converted to current structure level
 %RMU−F−RELMAXIDBAD, ROLLING BACK CONVERSION − Relation ID exceeds maximum
  8192 for system table RDB$RELATIONS
 %RMU−F−CVTROLSUC, CONVERT rolled−back for
  DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1 to version V7.0
```

The following example shows the normal case when the maximum allowed relation ID is not exceeded:

```
$rmu/convert mf_personnel
%RMU−I−RMUTXT_000, Executing RMU for Oracle Rdb V7.1−00
Are you satisfied with your backup of
 DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1 and your backup of
 any associated .aij files [N]? Y
%RMU−I−LOGCONVRT, database root converted to current structure level
%RMU−S−CVTDBSUC, database DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1
 successfully converted from version V7.0 to V7.1
%RMU−I−CVTCOMSUC, CONVERT committed for
 DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1 to version V7.1
```

7.3.3 RMU Unload /After_Journal Requires Accurate AIP Logical Area Information

The RMU Unload /After_Journal command uses the on−disk area inventory pages (AIPs) to determine the appropriate type of each logical area when reconstructing logical dbkeys for records stored in mixed−format storage areas. However, the logical area type information in the AIP is generally unknown for logical areas created prior to Oracle Rdb release 7.0.1. If the RMU Unload /After_Journal command cannot determine the logical area type for one or more AIP entries, a warning message is displayed for each such area and may ultimately return logical dbkeys with a 0 (zero) area number for records stored in mixed−format storage areas.

In order to update the on−disk logical area type in the AIP, the RMU Repair utility must be used. The INITIALIZE=LAREA_PARAMETERS=optionfile qualifier option file can be used with the TYPE qualifier. For example, to repair the EMPLOYEES table of the MF_PERSONNEL database, you would create an options file that contains the following line:

EMPLOYEES /TYPE=TABLE

For partitioned logical areas, the AREA=name qualifier can be used to identify the specific storage areas that are to be updated. For example, to repair the EMPLOYEES table of the MF_PERSONNEL database for the EMPID_OVER storage area only, you would create an options file that contains the following line:

EMPLOYEES /AREA=EMPID_OVER /TYPE=TABLE

The TYPE qualifier specifies the type of a logical area. The following keywords are allowed:

- TABLE
  Specifies that the logical area is a data table. This would be a table created using the SQL CREATE TABLE syntax.

- B−TREE
  Specifies that the logical area is a B−tree index. This would be an index created using the SQL CREATE INDEX TYPE IS SORTED syntax.
- HASH
  Specifies that the logical area is a hash index. This would be an index created using the SQL CREATE INDEX TYPE IS HASHED syntax.
- SYSTEM
  Specifies that the logical area is a system record that is used to identify hash buckets. Users cannot explicitly create these types of logical areas.

  Note

  This type should NOT be used for the RDB$SYSTEM logical areas. This type does NOT identify system relations.
- BLOB
  Specifies that the logical area is a BLOB repository.

There is no explicit error checking of the type specified for a logical area. However, an incorrect type may cause the RMU Unload /After_Journal command to be unable to correctly return valid, logical dbkeys.

7.3.4 Do Not Use HYPERSORT with RMU Optimize After_Journal Command

The OpenVMS Alpha V7.1 operating system introduced the high−performance Sort/Merge utility (also known as HYPERSORT). This utility takes advantage of the OpenVMS Alpha architecture to provide better performance for most sort and merge operations.

The high−performance Sort/Merge utility supports a subset of the SOR routines. Unfortunately, the high−performance Sort/Merge utility does not support several of the interfaces used by the RMU Optimize After_Journal command. In addition, the high−performance Sort/Merge utility reports no error or warning when being called with the unsupported options used by the RMU Optimize After_Journal command.

Because of this, the use of the high−performance Sort/Merge utility is not supported for the RMU Optimize After_Journal command. Do not define the logical name SORTSHR to reference HYPERSORT.EXE.

7.3.5 Changes in EXCLUDE and INCLUDE Qualifiers for RMU Backup

The RMU Backup command no longer accepts both the Include and Exclude qualifiers in the same command. This change removes the confusion over exactly what gets backed up when Include and Exclude are specified on the same line, but does not diminish the capabilities of the RMU Backup command.

To explicitly exclude some storage areas from a backup, use the Exclude qualifier to name the storage areas to be excluded. This causes all storage areas to be backed up except for those named by the Exclude qualifier.

Similarly, the Include qualifier causes only those storage areas named by the qualifier to be backed up. Any storage area not named by the Include qualifier is not backed up. The Noread_only and Noworm qualifiers continue to cause read−only storage areas and WORM storage areas to be omitted from the backup even if these areas are explicitly listed by the Include qualifier.

Another related change is in the behavior of EXCLUDE=*. In previous versions, EXCLUDE=* caused all storage areas to be backed up. Beginning with V7.1, EXCLUDE=* causes only a root backup to be done. A backup created by using EXCLUDE=* can be used only by the RMU Restore Only_Root command.

7.3.6 RMU Backup Operations Should Use Only One Type of Tape Drive

When using more than one tape drive for an RMU Backup command, all of the tape drives must be of the same type (for example, all the tape drives must be TA90s or TZ87s or TK50s). Using different tape drive types (for example, one TK50 and one TA90) for a single database backup operation may make database restoration difficult or impossible.

Oracle RMU attempts to prevent using different tape drive densities during a backup operation, but is not able to detect all invalid cases and expects that all tape drives for a backup are of the same type.

As long as all of the tapes used during a backup operation can be read by the same type of tape drive during a restore operation, the backup is likely valid. This may be the case, for example, when using a TA90 and a TA90E.

Oracle Corporation recommends that, on a regular basis, you test your backup and recovery procedures and environment using a test system. You should restore the database and then recover using AIJs to simulate failure recovery of the production system.

Consult the Oracle Rdb7 Guide to Database Maintenance, the Oracle Rdb7 Guide to Database Design and Definition, and the Oracle RMU Reference Manual for additional information about Oracle Rdb backup and restore operations.

7.3.7 RMU/VERIFY Reports PGSPAMENT or PGSPMCLST Errors

RMU/VERIFY may sometimes report PGSPAMENT or PGSPMCLST errors when verifying storage areas. These errors indicate that the Space Area Management (SPAM) page fullness threshold for a particular data page does not match the actual space usage on the data page. For a further discussion of SPAM pages, consult the Oracle Rdb7 Guide to Database Maintenance.

In general, these errors will not cause any adverse affect on the operation of the database. There is potential for space on the data page to not be totally utilized, or for a small amount of extra I/O to be expended when searching for space in which to store new rows. But unless there are many of these errors then the impact should be negligible.

It is possible for these inconsistencies to be introduced by errors in Oracle Rdb. When those cases are discovered, Oracle Rdb is corrected to prevent the introduction of the inconsistencies. It is also possible for these errors to be introduced during the normal operation of Oracle Rdb. The following scenario can leave the SPAM pages inconsistent:

1. A process inserts a row on a page, and updates the threshold entry on the corresponding SPAM page to reflect the new space utilization of the data page. The data page and SPAM pages are not flushed to disk.

2. Another process notifies the first process that it would like to access the SPAM page being held by the process. The first process flushes the SPAM page changes to disk and releases the page. Note that it has not flushed the data page.
3. The first process then terminates abnormally (for example, from the DCL STOP/IDENTIFICATION command). Since that process never flushed the data page to disk, it never wrote the changes to the Recovery Unit Journal (RUJ) file. Since there were no changes in the RUJ file for that data page then the Database Recovery (DBR) process did not need to roll back any changes to the page. The SPAM page retains the threshold update change made above even though the data page was never flushed to disk.

While it would be possible to create mechanisms to ensure that SPAM pages do not become out of synch with their corresponding data pages, the performance impact would not be trivial. Since these errors are relatively rare and the impact is not significant, then the introduction of these errors is considered to be part of the normal operation of Oracle Rdb. If it can be proven that the errors are not due to the scenario above, then Oracle Product Support should be contacted.

PGSPAMENT and PGSPMCLST errors may be corrected by doing any one of the following operations:

- Recreate the database by performing:
    1. SQL EXPORT
    2. SQL DROP DATABASE
    3. SQL IMPORT
- Recreate the database by performing:
    1. RMU/BACKUP
    2. SQL DROP DATABASE
    3. RMU/RESTORE
- Repair the SPAM pages by using the RMU/REPAIR command. Note that the RMU/REPAIR command does not write its changes to an after−image journal (AIJ) file. Therefore, Oracle recommends that a full database backup be performed immediately after using the RMU/REPAIR command.

7.4 Known Problems and Restrictions in All Interfaces for Release 7.0 and Earlier

The following problems and restrictions from release 7.0 and earlier still exist.

7.4.1 Converting Single−File Databases

Because of a substantial increase in the database root file information for V7.0, you should ensure that you have adequate disk space before you use the RMU Convert command with single−file databases and V7.0 or higher.

The size of the database root file of any given database increases a minimum of 13 blocks and a maximum of 597 blocks. The actual increase depends mostly on the maximum number of users specified for the database.

7.4.2 Row Caches and Exclusive Access

If a table has a row−level cache defined for it, the Row Cache Server (RCS) may acquire a shared lock on the table and prevent any other user from acquiring a Protective or Exclusive lock on that table.

7.4.3 Exclusive Access Transactions May Deadlock with RCS Process

If a table is frequently accessed by long running transactions that request READ/WRITE access reserving the table for EXCLUSIVE WRITE and if the table has one or more indexes, you may experience deadlocks between the user process and the Row Cache Server (RCS) process.

There are at least three suggested workarounds to this problem:

- Reserve the table for SHARED WRITE
- Close the database and disable row cache for the duration of the exclusive transaction
- Change the checkpoint interval for the RCS process to a time longer than the time required to complete the batch job and then trigger a checkpoint just before the batch job starts. Set the interval back to a smaller interval after the checkpoint completes.

7.4.4 Strict Partitioning May Scan Extra Partitions

When you use a WHERE clause with the less than (<) or greater than (>) operator and a value that is the same as the boundary value of a storage map, Oracle Rdb scans extra partitions. A boundary value is a value specified in the WITH LIMIT OF clause. The following example, executed while the logical name RDMS$DEBUG_FLAGS is defined as "S", illustrates the behavior:

```
ATTACH 'FILENAME MF_PERSONNEL';
CREATE TABLE T1 (ID INTEGER, LAST_NAME CHAR(12), FIRST_NAME CHAR(12));
CREATE STORAGE MAP M FOR T1 PARTITIONING NOT UPDATABLE
 STORE USING (ID)
 IN EMPIDS_LOW WITH LIMIT OF (200)
 IN EMPIDS_MID WITH LIMIT OF (400)
 OTHERWISE IN EMPIDS_OVER;
INSERT INTO T1 VALUES (150,'Boney','MaryJean');
INSERT INTO T1 VALUES (350,'Morley','Steven');
INSERT INTO T1 VALUES (300,'Martinez','Nancy');
INSERT INTO T1 VALUES (450,'Gentile','Russ');
SELECT * FROM T1 WHERE ID > 400;
Conjunct Get Retrieval sequentially of relation T1
Strict Partitioning: part 2 3
ID LAST_NAME FIRST_NAME
450 Gentile Russ
1 row selected
```

In the previous example, partition 2 does not need to be scanned. This does not affect the correctness of the result. Users can avoid the extra scan by using values other than the boundary values.

7.4.5 Restriction When Adding Storage Areas with Users Attached to Database

If you try to interactively add a new storage area where the page size is less than the existing page size and the database has been manually opened or users are active, the add operation fails with the following error:

```
%RDB−F−SYS_REQUEST, error from system services request
−RDMS−F−FILACCERR, error opening database root DKA0:[RDB]TEST.RDB;1
−SYSTEM−W−ACCONFLICT, file access conflict
```

You can make this change only when no users are attached to the database and, if the database is set to OPEN IS MANUAL, the database is closed. Several internal Oracle Rdb data structures are based on the minimum page size and these structures cannot be resized if users are attached to the database.

Furthermore, because this particular change is not recorded in the AIJ, any recovery scenario fails. Note also that if you use .aij files, you must backup the database and restart after−image journaling because this change invalidates the current AIJ recovery.

7.4.6 Support for Single−File Databases to Be Dropped in a Future Release

Oracle Rdb currently supports both single−file and multifile databases on all platforms. However, single−file databases will not be supported in a future release of Oracle Rdb. At that time, Oracle Rdb will provide the means to easily convert single−file databases to multifile databases.

Oracle Rdb recommends that users with single−file databases perform the following actions:

- Use the Oracle RMU commands, such as Backup and Restore, to make copies, backup, or move single−file databases. Do not use operating system commands to copy, back up, or move databases.
- Create new databases as multifile databases even though single−file databases are supported.

7.4.7 Multiblock Page Writes May Require Restore Operation

If a node fails while a multiblock page is being written to disk, the page in the disk becomes inconsistent, and is detected immediately during failover. (Failover is the recovery of an application by restarting it on another computer.) The problem is rare, and occurs because only single−block I/O operations are guaranteed by OpenVMS to be written atomically. This problem has never been reported by any customer and was detected only during stress tests in our labs.

Correct the page by an area−level restore operation. Database integrity is not compromised, but the affected area is not available until the restore operation completes.

A future release of Oracle Rdb will provide a solution that guarantees multiblock atomic write operations. Cluster failovers will automatically cause the recovery of multiblock pages, and no manual intervention will be required.

7.4.8 Replication Option Copy Processes Do Not Process Database Pages Ahead of an Application

When a group of copy processes initiated by the Replication Option (formerly Data Distributor) begins running after an application has begun modifying the database, the copy processes catch up to the application and are not able to process database pages that are logically ahead of the application in the RDB$CHANGES system relation. The copy processes all align waiting for the same database page and do not move on until the application has released it. The performance of each copy process degrades because it is being paced by the application.

When a copy process completes updates to its respective remote database, it updates the RDB$TRANSFERS system relation and then tries to delete any RDB$CHANGES rows not needed by any transfers. During this process, the RDB$CHANGES table cannot be updated by any application process, holding up any database updates until the deletion process is complete. The application stalls while waiting for the RDB$CHANGES table. The resulting contention for RDB$CHANGES SPAM pages and data pages severely impacts performance throughput, requiring user intervention with normal processing.

This is a known restriction in V4.0 and higher. Oracle Rdb uses page locks as latches. These latches are held only for the duration of an action on the page and not to the end of transaction. The page locks also have blocking asynchronous system traps (ASTs) associated with them. Therefore, whenever a process requests a page lock, the process holding that page lock is sent a blocking AST (BLAST) by OpenVMS. The process that receives such a blocking AST queues the fact that the page lock should be released as soon as possible. However, the page lock cannot be released immediately.

Such work requests to release page locks are handled at verb commit time. An Oracle Rdb verb is an Oracle Rdb query that executes atomically, within a transaction. Therefore, verbs that require the scan of a large table, for example, can be quite long. An updating application does not release page locks until its verb has completed.

The reasons for holding on to the page locks until the end of the verb are fundamental to the database management system.

7.5 SQL Known Problems and Restrictions for Oracle Rdb Release 7.0 and Earlier

The following problems and restrictions from Oracle Rdb Release 7.0 and earlier still exist.

7.5.1 SQL Does Not Display Storage Map Definition After Cascading Delete of Storage Area

When you drop a storage area using the CASCADE keyword and that storage area is not the only area to which the storage map refers, the SHOW STORAGE MAP statement no longer shows the placement definition for that storage map.

The following example demonstrates this restriction:

```
SQL> SHOW STORAGE MAP DEGREES_MAP1
     DEGREES_MAP1
 For Table:             DEGREES1
 Compression is:        ENABLED
 Partitioning is:       NOT UPDATABLE
 Store clause:          STORE USING (EMPLOYEE_ID)
            IN DEG_AREA WITH LIMIT OF ('00250')
             OTHERWISE IN DEG_AREA2

SQL> DISCONNECT DEFAULT;
SQL> −− Drop the storage area, using the CASCADE keyword.
SQL> ALTER DATABASE FILENAME MF_PERSONNEL
cont> DROP STORAGE AREA DEG_AREA CASCADE;
SQL> −− Display the storage map definition.
SQL> ATTACH 'FILENAME MF_PERSONNEL';
SQL> SHOW STORAGE MAP DEGREES_MAP1
DEGREES_MAP1 For Table: DEGREES1
Compression is: ENABLED
Partitioning is: NOT UPDATABLE
```

The other storage area, DEG_AREA2, still exists, even though the SHOW STORAGE MAP statement does not display it.

A workaround is to use the RMU Extract command with the Items=Storage_Map qualifier to see the mapping.

7.5.2 ARITH_EXCEPT or Incorrect Results Using LIKE IGNORE CASE

When you use LIKE...IGNORE CASE, programs linked under Oracle Rdb V4.2 and V5.1, but run under higher versions of Oracle Rdb, may result in incorrect results or %RDB−E−ARITH_EXCEPT exceptions.

To work around the problem, avoid using IGNORE CASE with LIKE or recompile and relink under a higher version (V6.0 or higher.)

7.5.3 Different Methods of Limiting Returned Rows from Queries

You can establish the query governor for rows returned from a query by using either the SQL SET QUERY LIMIT statement or a logical name. This note describes the differences between the two mechanisms.

If you define the RDMS$BIND_QG_REC_LIMIT logical name to a small value, the query often fails with no rows returned regardless of the value assigned to the logical. The following example demonstrates setting the limit to 10 rows and the resulting failure:

```
$ DEFINE RDMS$BIND_QG_REC_LIMIT 10
$ SQL$
SQL> ATTACH 'FILENAME MF_PERSONNEL';
SQL> SELECT EMPLOYEE_ID FROM EMPLOYEES;
%RDB−F−EXQUOTA, Oracle Rdb runtime quota exceeded
−RDMS−E−MAXRECLIM, query governor maximum limit of rows has been reached
```

Interactive SQL must load its metadata cache for the table before it can process the SELECT statement. In this example, interactive SQL loads its metadata cache to allow it to check that the column EMPLOYEE_ID really exists for the table. The queries on the Oracle Rdb system relations RDB$RELATIONS and RDB$RELATION_FIELDS exceed the limit of rows.

Oracle Rdb does not prepare the SELECT statement, let alone execute it. Raising the limit to a number less than 100 (the cardinality of EMPLOYEES) but more than the number of columns in EMPLOYEES (that is, the number of rows to read from the RDB$RELATION_FIELDS system relation) is sufficient to read each column definition.

To see an indication of the queries executed against the system relations, define the RDMS$DEBUG_FLAGS logical name as "S" or "B".

If you set the row limit using the SQL SET QUERY statement and run the same query, it returns the number of rows specified by the SQL SET QUERY statement before failing:

```
 SQL> ATTACH 'FILENAME MF_PERSONNEL';
 SQL> SET QUERY LIMIT ROWS 10;
 SQL> SELECT EMPLOYEE_ID FROM EMPLOYEES;
 EMPLOYEE_ID
 00164
 00165
    .
    .
    .
 00173
 %RDB−E−EXQUOTA, Oracle Rdb runtime quota exceeded
 −RDMS−E−MAXRECLIM, query governor maximum limit of rows has been reached
```

The SET QUERY LIMIT specifies that only user queries be limited to 10 rows. Therefore, the queries used to load the metadata cache are not restricted in any way.

Like the SET QUERY LIMIT statement, the SQL precompiler and module processor command line qualifiers (QUERY_MAX_ROWS and SQLOPTIONS=QUERY_MAX_ROWS) only limit user queries.

Keep the differences in mind when limiting returned rows using the logical name RDMS$BIND_QG_REC_LIMIT. They may limit more queries than are obvious. This is important when using 4GL tools, the SQL precompiler, the SQL module processor, and other interfaces that read the Oracle Rdb system relations as part of query processing.

7.5.4 Suggestions for Optimal Use of SHARED DATA DEFINITION Clause for Parallel Index Creation

The CREATE INDEX process involves the following steps:

1. Process the metadata.
2. Lock the index name.
   Because new metadata (which includes the index name) is not written to disk until the end of the index process, Oracle Rdb must ensure index name uniqueness across the database during this time by taking a special lock on the provided index name.
3. Read the table for sorting by selected index columns and ordering.
4. Sort the key data.
5. Build the index (includes partitioning across storage areas).
6. Write new metadata to disk.

Step 6 is the point of conflict with other index definers because the system relation and indexes are locked like any other updated table.

Multiple users can create indexes on the same table by using the RESERVING table_name FOR SHARED DATA DEFINITION clause of the SET TRANSACTION statement. For optimal usage of this capability, Oracle Rdb suggests the following guidelines:

- You should commit the transaction immediately after the CREATE INDEX statement so that locks on the table are released. This avoids lock conflicts with other index definers and improves overall concurrency.
- By assigning the location of the temporary sort work files SORTWORK0, SORTWORK1, ... , SORTWORK9 to different disks for each parallel process that issues the SHARED DATA DEFINITION statement, you can increase the efficiency of sort operations. This minimizes any possible disk I/O bottlenecks and allows overlap of the SORT read/write cycle.
- If possible, enable global buffers and specify a buffer number large enough to hold a sufficient amount of table data. However, do not define global buffers larger than the available system physical memory. Global buffers allow sharing of database pages and thus result in disk I/O savings. That is, pages are read from disk by one of the processes and then shared by the other index definers for the same table, reducing the I/O load on the table.
- If global buffers are not used, ensure that enough local buffers exist to keep much of the index cached (use the RDM$BIND_BUFFERS logical name or the NUMBER OF BUFFERS IS clause in SQL to change the number of buffers).
- To distribute the disk I/O load, store the storage areas for the indexes on separate disk drives. Note that using the same storage area for multiple indexes results in contention during the index creation (Step 5) for SPAM pages.
- Consider placing the .ruj file for each parallel definer on its own disk or an infrequently used disk.
- Even though snapshot I/O should be minimal, consider disabling snapshots during parallel index creation.
- Refer to the Oracle Rdb7 Guide to Database Performance and Tuning to determine the appropriate working set values for each process to minimize excessive paging activity. In

particular, avoid using working set parameters where the difference between WSQUOTA and WSEXTENT is large. The SORT utility uses the difference between these two values to allocate scratch virtual memory. A large difference (that is, the requested virtual memory grossly exceeds the available physical memory) may lead to excessive page faulting.

- ♦ The performance benefits of using SHARED DATA DEFINITION can best be observed when creating many indexes in parallel. The benefit is in the average elapsed time, not in CPU or I/O usage. For example, when two indexes are created in parallel using the SHARED DATA DEFINITION clause, the database must be attached twice, and the two attaches each use separate system resources.
- ♦ Using the SHARED DATA DEFINITION clause on a single−file database or for indexes defined in the RDB$SYSTEM storage area is not recommended.

The following table displays the elapsed time benefit when creating multiple indexes in parallel with the SHARED DATA DEFINITION clause. The table shows the elapsed time for ten parallel process index creations (Index1, Index2, ... Index10) and one process with ten sequential index creations (All10). In this example, global buffers are enabled and the number of buffers is 500. The longest time for a parallel index creation is Index7 with an elapsed time of 00:02:34.64, compared to creating ten indexes sequentially with an elapsed time of 00:03:26.66. The longest single parallel create index elapsed time is shorter than the elapsed time of creating all ten of the indexes serially.

Table 7−2 Elapsed Time for Index Creations

Index Create Job	Elapsed Time
Index1	00:02:22.50
Index2	00:01:57.94
Index3	00:02:06.27
Index4	00:01:34.53
Index5	00:01:51.96
Index6	00:01:27.57
Index7	00:02:34.64
Index8	00:01:40.56
Index9	00:01:34.43
Index10	00:01:47.44
All10	00:03:26.66

7.5.5 Side Effect When Calling Stored Routines

When calling a stored routine, you must not use the same routine to calculate argument values by a stored function. For example, if the routine being called is also called by a stored function during the calculation of an argument value, passed arguments to the routine may be incorrect.

The following example shows a stored procedure P being called during the calculation of the arguments for another invocation of the stored procedure P:

```
SQL> create module M
cont>     language SQL
cont>
cont>     procedure P (in :a integer, in :b integer, out :c integer);
cont>     begin
cont>     set :c = :a + :b;
cont>     end;
cont>
cont>     function F () returns integer
cont>     comment is 'expect F to always return 2';
cont>     begin
cont>     declare :b integer;
cont>     call P (1, 1, :b);
cont>     trace 'returning ', :b;
cont>     return :b;
cont>     end;
cont> end module;
SQL>
SQL> set flags 'TRACE';
SQL> begin
cont> declare :cc integer;
cont> call P (2, F(), :cc);
cont> trace 'Expected 4, got ', :cc;
cont> end;
~Xt: returning 2
~Xt: Expected 4, got 3
```

The result as shown above is incorrect. The routine argument values are written to the called routine's parameter area before complex expression values are calculated. These calculations may (as in the example) overwrite previously copied data.

The workaround is to assign the argument expression (in this example calling the stored function F) to a temporary variable and pass this variable as the input for the routine. The following example shows the workaround:

```
SQL> begin
cont> declare :bb, :cc integer;
cont> set :bb = F();
cont> call P (2, :bb, :cc);
cont> trace 'Expected 4, got ', :cc;
cont> end;
~Xt: returning 2
~Xt: Expected 4, got 4
```

This problem will be corrected in a future version of Oracle Rdb.

7.5.6 Considerations When Using Holdable Cursors

If your applications use holdable cursors, be aware that after a COMMIT or ROLLBACK statement is executed, the result set selected by the cursor may not remain stable. That is, rows may be inserted, updated, and deleted by other users because no locks are held on the rows selected by the holdable cursor after a commit or rollback occurs. Moreover, depending on the access strategy, rows not yet fetched may change before Oracle Rdb actually fetches them.

As a result, you may see the following anomalies when using holdable cursors in a concurrent user environment:

♦ If the access strategy forces Oracle Rdb to take a data snapshot, the data read and cached may

be stale by the time the cursor fetches the data.
For example, user 1 opens a cursor and commits the transaction. User 2 deletes rows read by user 1 (this is possible because the read locks are released). It is possible for user 1 to report data now deleted and committed.

- If the access strategy uses indexes that allow duplicates, updates to the duplicates chain may cause rows to be skipped, or even revisited.
  Oracle Rdb keeps track of the dbkey in the duplicate chain pointing to the data that was fetched. However, the duplicates chain could be revised by the time Oracle Rdb returns to using it.

Holdable cursors are a very powerful feature for read−only or predominantly read−only environments. However, in concurrent update environments, the instability of the cursor may not be acceptable. The stability of holdable cursors for update environments will be addressed in future versions of Oracle Rdb.

You can define the logical name RDMS$BIND_HOLD_CURSOR_SNAP to the value 1 to force all hold cursors to fetch the result set into a cached data area. (The cached data area appears as a "Temporary Relation" in the optimizer strategy displayed by the SET FLAGS 'STRATEGY' statement or the RDMS$DEBUG_FLAGS "S" flag.) This logical name helps to stabilize the cursor to some degree.

| Contents