# Oracle Rdb7™

# SQL Reference Manual
# Volume 2

Release 7.0

Part No. A42813-1

ORACLE®

SQL Reference Manual, Volume 2

Release 7.0

Part No. A42813-1

# Contents

# Send Us Your Comments

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

You can send comments to us in the following ways:

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

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

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

| | |
|---|---|
| Name ______________________ | Title ______________________ |
| Company ______________________ | Department ______________________ |
| Mailing Address ______________________ | Telephone Number ______________________ |
| ______________________ | |
| Book Title ______________________ | Version Number ______________________ |

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

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

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

# Preface

This manual describes the syntax and semantics of all the statements and language elements for the SQL (structured query language) interface to the Oracle Rdb database software.

## Intended Audience

To get the most out of this manual, you should be familiar with data processing procedures, basic database management concepts and terminology, and the OpenVMS operating system.

## Operating System Information

You can find information about the versions of the operating system and optional software that are compatible with this version of Oracle Rdb in the *Oracle Rdb7 Installation and Configuration Guide*.

For information on the compatibility of other software products with this version of Oracle Rdb, refer to the *Oracle Rdb7 Release Notes*.

Contact your Oracle representative if you have questions about the compatibility of other software products with this version of Oracle Rdb.

## Structure

This manual is divided into three volumes. Volume 1 contains Chapter 1 through Chapter 5 and an index. Volume 2 contains Chapter 6 and an index. Volume 3 contains Chapter 7, the appendixes, and an index.

The index for each volume contains entries for the respective volume only and does not contain index entries from the other volumes in the set.

The following table shows the contents of the chapters and appendixes in Volumes 1, 2, and 3 of the *Oracle Rdb7 SQL Reference Manual*:

| | |
|---|---|
| Chapter 1 | Introduces SQL (structured query language) and briefly describes SQL functions. This chapter also describes conformance to the ANSI standard, how to read syntax diagrams, executable and nonexecutable statements, keywords and line terminators, and support for Multivendor Integration Architecture. |
| Chapter 2 | Describes the language and syntax elements common to many SQL statements. |
| Chapter 3 | Describes the syntax for the SQL module language and the SQL module processor command line. |
| Chapter 4 | Describes the syntax of the SQL precompiler command line. |
| Chapter 5 | Describes SQL routines. |
| Chapter 6 and Chapter 7 | Describes in detail the syntax and semantics of the SQL statements. These chapters include descriptions of data definition statements, data manipulation statements, and interactive control commands. |
| Appendix A | Describes the different types of errors encountered in SQL and where they are documented. |
| Appendix B | Describes the SQL Communications Area and the message vector. |
| Appendix C | Describes the SQLSTATE error handling mechanism. |
| Appendix D | Describes the SQL Descriptor Areas and how they are used in dynamic SQL programs. |
| Appendix E | Summarizes the logical names and configuration parameters that SQL recognizes for special purposes. |
| Appendix F | Summarizes the obsolete SQL features of the current Oracle Rdb version. |
| Appendix G | Summarizes the SQL functions that have been added to the Oracle Rdb SQL interface for convergence with Oracle7 SQL. |
| Index | Volume 2 only. |

## Related Manuals

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

- *Oracle Rdb7 Guide to Database Design and Definition*
- *Oracle Rdb7 Guide to Database Performance and Tuning*
- *Oracle Rdb7 Introduction to SQL*
- *Oracle Rdb7 Guide to SQL Programming*

## Conventions

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

| | |
|---|---|
| Digital UNIX | This icon denotes the beginning of information specific to the Digital UNIX operating system. |
| OpenVMS VAX OpenVMS Alpha | This icon combination denotes the beginning of information specific to both the OpenVMS VAX and OpenVMS Alpha operating systems. |
| ♦ | The diamond symbol denotes the end of a section of information specific to an operating system or platform. |

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

Often in examples the prompts are not shown. Generally, they are shown where it is important to depict an interactive sequence exactly; otherwise, they are omitted.

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

The following conventions are also used in this manual:

| | |
|---|---|
| .<br>.<br>. | Vertical ellipsis points in an example mean that information not directly related to the example has been omitted. |
| . . . | Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted. |
| e, f, t | Index entries in the printed manual may have a lowercase e, f, or t following the page number; the e, f, or t is a reference to the example, figure, or table, respectively, on that page. |
| **boldface text** | Boldface type in text indicates a new term. |
| < > | Angle brackets enclose user-supplied names in syntax diagrams. |

| | |
|---|---|
| [ ] | Brackets enclose optional clauses from which you can choose one or none. |
| $ | The dollar sign represents the command language prompt. This symbol indicates that the command language interpreter is ready for input. |
| UPPERCASE<br>lowercase | The Digital UNIX operating system differentiates between lowercase and uppercase characters. Examples, syntax descriptions, function definitions, and literal strings that appear in text must be typed exactly as shown. |

## References to Products

The Oracle Rdb documentation set to which this manual belongs often refers to the following Oracle Corporation products by their abbreviated names:

- In this manual, Oracle Rdb refers to Oracle Rdb for OpenVMS and Oracle Rdb for Digital UNIX software. Version 7.0 of Oracle Rdb software is often referred to as V7.0.
- The SQL interface to Oracle Rdb is referred to as SQL. This interface is the Oracle Rdb implementation of the SQL standard ANSI X3.135-1992, ISO 9075:1992, commonly referred to as the ANSI/ISO SQL standard or SQL92.
- Oracle CDD/Repository software is referred to as the dictionary, the data dictionary, or the repository.
- Oracle ODBC Driver for Rdb software is referred to as the ODBC driver.
- OpenVMS means both the OpenVMS Alpha and OpenVMS VAX operating system.

# Technical Changes and New Features

This section identifies the new and updated portions of this manual since it was last released with V6.0.

The *Oracle Rdb7 Release Notes* describes current limitations and restrictions.

**The major new features and technical changes for V6.1 that are described in this manual are:**

- INTEGER data type for SQL module language allows modifiers

  The SQL module language syntax has been extended to allow specification of precise INTEGER module parameters in the number of bits.

- New command line qualifiers for SQL module language and precompiled SQL

  Table 1 shows the new qualifiers for SQL module language and precompiled SQL and the appropriate platform.

**Table 1 Command Line Qualifiers**

| Qualifier Name | Digital UNIX | OpenVMS Alpha | OpenVMS VAX |
|---|---|---|---|
| **SQL Module Language** | | | |
| [NO]ALIGN_RECORDS | | X | X |
| –[no]align | X | | |
| [NO]LOWERCASE_PROCEDURE_ NAMES | | X | X |
| –[no]lc_proc | X | | |
| [NO]C_PROTOTYPES | | X | X |
| –[no]cproto | X | | |

(continued on next page)

**Table 1 (Cont.) Command Line Qualifiers**

| Qualifier Name | Digital UNIX | OpenVMS Alpha | OpenVMS VAX |
|---|---|---|---|
| **SQL Module Language** | | | |
| [NO]LONG_SQLCODE | | X | X |
| –[no]lsqlcode | X | | |
| [NO]EXTERNAL_GLOBALS | | X | X |
| –[no]extern | X | | |
| USER_DEFAULT | | X | X |
| –user username | X | | |
| PASSWORD_DEFAULT | | X | X |
| –pass password | X | | |
| [NO]PACKAGE_COMPILATION | | X | X |
| ROLLBACK_ON_EXIT | | X | X |
| –fida | X | | |
| –int32 | X | | |
| –int64 | X | | |
| –plan file-spec | X | | |
| **SQL Precompiler** | | | |
| [NO]DECLARE_MESSAGE_ VECTOR | | X | X |
| –s ′–[no]msgvec′ | X | | |
| USER_DEFAULT | | X | X |
| –s ′–user username′ | X | | |
| PASSWORD_DEFAULT | | X | X |
| –s ′–pass password′ | X | | |
| ROLLBACK_ON_EXIT | | X | X |
| [NO]EXTERNAL_GLOBALS | | X | X |
| –s ′–[no]extern′ | X | | |
| –plan file-spec | X | | |

See Chapter 3 and Chapter 4 for more information.

- Asynchronous creation of storage areas

You can specify whether Oracle Rdb creates storage areas serially, creates a specified number at the same time, or creates all areas at the same time.

For information about the SQL syntax, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Authenticating users for remote access

  Oracle Rdb lets you explicitly provide user name and password information in SQL statements that attach to the database. In addition, it lets you pass the information to an SQL module language or precompiled SQL program by using a parameter and new command line qualifiers. You can also pass the information to Oracle Rdb by using configuration parameters.

- Selecting an outline to use for a query

  Using SQL syntax, you can specify the name of an outline to use for a query.

  SQL statements affected by this feature are DECLARE CURSOR, DELETE, INSERT, SELECT, and UPDATE and select expression.

OpenVMS VAX OpenVMS Alpha

- Notification of classes of operators

  Using SQL syntax, you can specify which classes of operators are notified in the case of a catastrophic journaling event such as running out of disk space. (This feature was available in V6.0 using the RMU interface.)

  For information about the SQL syntax, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement. ♦

- Specifying shutdown time

  Using SQL syntax, you can specify the number of minutes the database system will wait after a catastrophic event before it shuts down the database. (This feature was available in V6.0 using the RMU interface.)

  For information about the SQL syntax, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Asynchronous batch-writes

  Using SQL syntax, you can specify that processes write batches of modified data pages to disk asynchronously (the process does not stall while waiting for the batch-write operation to complete). Asynchronous batch-writes improve the performance of update applications without the loss of data integrity. (This feature was available in V6.0 using logical names to specify the number of buffers used.)

For information about the SQL syntax, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Asynchronous prefetch

  Using SQL syntax, you can specify whether or not Oracle Rdb reduces the amount of time that a process waits for pages to be read from disk by fetching pages before a process actually requests the pages.

  For information about the SQL syntax, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Fast incremental backup

  Using SQL syntax, you can specify whether Oracle Rdb checks each area's SPAM pages or each database page to find changes during incremental backup.

  For information about the SQL syntax, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Support for two new character sets

  Oracle Rdb includes support for two new character sets: BIG5 and TACTIS. BIG5 is a fixed 2-octet character set. TACTIS is a single-octet character set.

- TRIM built-in function

  The TRIM built-in function lets you remove leading and trailing characters from a character string.

- POSITION built-in function

  The POSITION built-in function lets you search for a particular substring within another string.

OpenVMS VAX OpenVMS Alpha

- INTEGRATE statement has new arguments

  Oracle Rdb provides a finer level of definition integration between an Oracle Rdb database and the CDD/Repository with the introduction of the DOMAIN and TABLE arguments to the INTEGRATE statement. In previous versions of Oracle Rdb, the INTEGRATE statement let you integrate all Oracle Rdb database schema objects with the CDD/Repository but did not allow the integration of individual schema objects. With Oracle Rdb V6.1, the INTEGRATE statement lets you select specific Oracle Rdb schema objects (tables and domains) for integration. However, SQL continues to let you integrate an entire database with the INTEGRATE statement when that level of integration is required. ♦

- SHOW DATABASE statement includes new information

  The output from the SHOW DATABASE statement includes information about the new database attributes, such as asynchronous batch-writes and shutdown time.

- LIKE predicate optimization in SQL queries

  Oracle Rdb has improved the performance of certain types of LIKE predicates in SQL queries.

- Multistring comments

  You can now specify comments that contain more than one string literal separated by a slash mark (/). This was implemented as a workaround to the limitation that comments can only be 1,024 characters in length. Statements affected by this new feature are:

  – COMMENT ON Statement

  – CREATE COLLATING SEQUENCE Statement

  – CREATE DATABASE Statement

  – CREATE FUNCTION Statement

  – CREATE MODULE Statement

  – CREATE OUTLINE Statement

  – CREATE PROCEDURE Statement

- New UNDECLARE Variable Statement

  You can now undeclare variables. See the UNDECLARE Variable Statement for more information.

- Three logical names introduced in Oracle Rdb V6.0 are deprecated and replaced with new names in V6.1. Table 2 shows the changes

**Table 2 Logical Name Changes**

| V6.0 OpenVMS Logical Name | V6.1 OpenVMS Logical Name | V6.1 Digital UNIX Configuration Parameter |
|---|---|---|
| RDM$BIND_ABW_DISABLED | RDM$BIND_ABW_ENABLED | RDB_BIND_ABW_ENABLED |
| RDM$BIND_APF_DISABLED | RDM$BIND_APF_ENABLED | RDB_BIND_APF_ENABLED |
| RDM$BIND_STATS_DISABLED | RDM$BIND_STATS_ENABLED | RDB_BIND_STATS_ENABLED |

SQL syntax has been introduced in Oracle Rdb V6.1 for these features. Oracle Rdb recommends that you use the SQL syntax for these features. See CREATE DATABASE Statement and ALTER DATABASE Statement for more information regarding the new syntax.

See Appendix E for more information regarding the new logical names.

- Portable SQL routines

  SQL provides the following routines for use on both OpenVMS and Digital UNIX operating systems. For more information, see the Routines topic under help for interactive SQL.

  - sql_close_cursors

    This routine closes all cursors. It functions the same as the SQL$CLOSE_CURSORS routine, which is available only on OpenVMS.

    On Digital UNIX, this routine is case sensitive and must be entered in lowercase.

  - sql_get_error_text

    This routine passes error text with formatted output to programs for processing. It is similar to the SQL$GET_ERROR_TEXT routine, which is available only on OpenVMS systems.

  - sql_get_message_vector

    This routine retrieves information from the message vector about the status of the last SQL statement.

    On Digital UNIX, this routine is case sensitive and must be entered in lowercase.

  - sql_get_error_handler, sql_register_error_handler, and sql_deregister_error_handler

    These routines now work on Digital UNIX, but otherwise have not changed from previous versions of Oracle Rdb.

  - sql_signal

This routine signals that an error has occurred on the execution of an SQL statement. It is equivalent to the SQL$SIGNAL routine, which is available only on OpenVMS systems.

Digital UNIX

- On Digital UNIX, the GLOBAL and EXTERNAL options of the DECLARE ALIAS statement differ.
  - GLOBAL

    Defines the alias to be globally visible
  - EXTERNAL

    Declares an external reference of the alias ♦

**The major new features and technical changes for V7.0 of Oracle Rdb that are described in this manual are:**

- Ranked B-tree structure

  Oracle Rdb now supports a new ranked B-tree structure that allows better optimization of queries, particularly queries involving range retrievals. Oracle Rdb is able to make better estimates of cardinality, reducing disk I/O and lock contention. To create a ranked B-tree structure, use the RANKED keyword of the CREATE INDEX . . . TYPE IS SORTED statement.

  A sorted ranked index allows storage of many records in a small space when you compress duplicates, using the DUPLICATES ARE COMPRESSED clause of the CREATE INDEX statement.

  For additional information, see the CREATE INDEX Statement.

OpenVMS Alpha

- System space global buffers

  Oracle Rdb for OpenVMS Alpha provides a new type of global buffer called system space buffers (SSB). The system space global buffer is located in the OpenVMS Alpha system space, which means that a system space global buffer is fully resident in memory and does not affect the quotas of the working set of the process. As a result, a process referencing a system space global buffer has an additional 256Mb of resident working set space.

  You can specify whether database root global buffers are created in system space or process space by using the SHARED MEMORY clause.

  See the ALTER DATABASE Statement, the CREATE CACHE Clause, the CREATE DATABASE Statement, and the IMPORT Statement for more information. ♦

OpenVMS Alpha

- Specifying if large memory is used to manage the row cache

  The LARGE MEMORY clause specifies if large memory is used to manage the row cache. Large memory allows Oracle Rdb to use as much physical memory as is available and to dynamically map it to the virtual address space of database users. It provides access to a large amount of physical memory through small virtual address windows.

  See the ALTER DATABASE Statement and the CREATE CACHE Clause for more information. ♦

- Row-level memory cache

  The row-level memory cache feature allows frequently referenced rows to remain in memory even when the associated page has been flushed back to disk. This saves in memory usage because only the more recently referenced rows are cached versus caching the entire buffer.

  See the CREATE CACHE Clause, the ALTER DATABASE Statement, the CREATE DATABASE Statement, the CREATE STORAGE AREA Clause, and the IMPORT Statement for more information regarding the row cache areas.

OpenVMS Alpha

- Specifying the number of window panes used by the large memory mapping algorithm

  See the ALTER DATABASE Statement and the CREATE CACHE Clause for more information. ♦

- Specifying if Oracle Rdb replaces rows in the cache when it becomes full

  See the ALTER DATABASE Statement and the CREATE CACHE Clause for more information.

- Specifying the FROM clause in the CREATE OUTLINE statement

  The process for creating outlines has been simplified with the new FROM syntax. You can now specify the statement for which you need an outline within the CREATE OUTLINE statement.

  See the CREATE OUTLINE Statement for more information.

- Freezing data definition changes

  You can ensure that the data definition of your database does not change by using the METADATA CHANGES ARE DISABLED clause of the ALTER DATABASE, CREATE DATABASE, or IMPORT statements.

  See the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement for more information regarding freezing data definition changes.

- Modifying the database buffer size

  You can now modify the database buffer size by using the BUFFER SIZE clause in the ALTER DATABASE statement. In previous versions, you could specify the clause only in the CREATE DATABASE statement.

  See the ALTER DATABASE Statement for more information regarding modifying the database buffer size.

- Creating a default storage area

  You can separate user data from the system data, such as the system tables, by using the DEFAULT STORAGE AREA clause of the CREATE DATABASE or IMPORT statements. This clause specifies that all user data and indexes that are not mapped explicitly to a storage area are stored in the default storage area.

  See the CREATE DATABASE Statement and the IMPORT Statement for more information regarding the default storage area.

- Deleting a storage area with a cascading delete

  You can specify that Oracle Rdb delete a storage area with a cascading delete. When you do, Oracle Rdb deletes database objects referring to the storage area.

  For more information, see the ALTER DATABASE Statement.

- Specifying how a database opens when you create the database

  You can specify whether a database opens automatically or manually when you create the database. In previous versions, you could specify the OPEN IS clause only in the ALTER DATABASE statement.

  See the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement for more information.

- Specifying how long to wait before closing a database

  You can specify how long Oracle Rdb waits before closing the database, by using the WAIT n MINUTES FOR CLOSE clause.

  See the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement for more information.

- Extending the allocation of storage areas

  You can now manually force the storage area to extend by using the ALLOCATION IS clause of the alter-storage-area-params clause.

  See the ALTER DATABASE Statement for more information.

- Vertical partitioning

  You can now partition a table vertically as well as horizontally. When you partition a table horizontally, you divide the rows of the table among storage areas according to data values in one or more columns. A given storage area then contains only those rows whose column values fall within the range that you specify. When you partition a table vertically, you divide the columns of the table among storage areas. A given storage area then contains only some of the columns of a table. Consider using vertical partitioning when you know that access to some of the columns in a table is frequent, but that the access to other columns is occasional.

  For more information, see the CREATE STORAGE MAP Statement.

- Strict partitioning

  You can now specify whether a partitioning key for a storage map is updatable or not updatable. If you specify that the key is not updatable, Oracle Rdb retrieval performance improves because Oracle Rdb can use the partitioning criteria when optimizing the query.

  For more information, see the CREATE STORAGE MAP Statement.

- Quickly deleting data in tables

  If you want to quickly delete the data in a table, but you want to maintain the metadata definition of the table (perhaps to reload the data into a new partitioning scheme), you can use the TRUNCATE TABLE statement.

  For more information, see the TRUNCATE TABLE Statement.

- Creating temporary tables

  You can create temporary tables to store temporary results only for a short duration, perhaps to temporarily store the results of a query so that your application can act on the results of that query. The data in a temporary table is deleted at the end of an SQL session.

  For more information, see the CREATE MODULE Statement, the CREATE TABLE Statement, and the DECLARE LOCAL TEMPORARY TABLE Statement.

- Removing the links with the repository

  You can remove the link between the repository and database but still maintain the data definitions in both places, using the DICTIONARY IS NOT USED clause of the ALTER DATABASE statement.

  For more information, see the ALTER DATABASE Statement.

- Specifying the location of the recovery journal file

  You can specify the location of the recovery journal using the RECOVERY JOURNAL (LOCATION IS 'directory-spec') clause when you alter, create, or import a database.

  For more information, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Specifying an edit string in an .aij backup file name

  You can specify if the backup file name includes an edit string with the EDIT STRING clause of the ALTER DATABASE statement.

  For more information, see the ALTER DATABASE Statement.

- Increasing the fanout factor for adjustable lock granularity

  Adjustable lock granularity for previous versions of Oracle Rdb defaulted to a count of 3. This means that the lock fanout factor was (10, 100, 1000). As databases grow larger, it is becoming necessary to allow these fanout factors to grow to reduce lock requirements for long queries. You can now change the fanout factor by specifying the COUNT IS clause with the ADJUSTABLE LOCK GRANULARITY IS ENABLED clause.

  For more information, the see ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Collecting a workload profile

  A workload profile is a description of the interesting table and column references used by queries in a database work load. When workload collection is enabled, the optimizer collects and records these references in the RDB$WORKLOAD system table.

  For more information, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Collecting cardinality updates

  When cardinality collection is enabled, the optimizer collects cardinalities for the table and non-unique indexes as rows are inserted or deleted from tables. The cardinalities are stored in the RDB$CARDINALITY column of the RDB$RELATIONS, RDB$INDICES, and RDB$INDEX_SEGMENTS system tables. Cardinality collection is enabled by default.

  For more information, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Specifying detected asynchronous prefetch with a threshold value

  Detected asynchronous prefetch can significantly improve performance by using heuristics to determine if an I/O pattern is sequential in behavior even if not actually performing sequential I/O. For example, when fetching a LIST OF BYTE VARYING column, the heuristics detect that the pages being fetched are sequential and fetch ahead asynchronously to avoid wait times when the page is really needed.

  For more information, see the ALTER DATABASE Statement, the CREATE DATABASE Statement, and the IMPORT Statement.

- Setting debug flags using SQL

  A new SET FLAGS statement has been added to interactive and dynamic SQL, and a SHOW FLAGS statement to interactive SQL. The new SET FLAGS statements has been added to enable and disable the database systems debug flags during execution. For more information, see the SET FLAGS Statement and the SHOW Statement.

- Cursors can now stay open across transactions (holdable cursors)

  SQL cursors can now remain open across transaction boundaries. The WITH HOLD clause of the DECLARE CURSOR statement indicates that the cursor will remain open after the transaction ends. A holdable cursor that has been held open retains its position when a new SQL transaction is begun.

  You can also specify the attributes of the holdable cursor as a database default using the SET HOLD CURSORS statement.

  For more information, see the DECLARE CURSOR Statement and the SET HOLD CURSORS Statement.

- External routine enhancements

  Starting with V7.0, external routines can now contain SQL statements to bind to new schema instances and perform database operations. External routine activation, execution, and exception handling is controlled by a new executor manager process.

  External routines are external functions or external procedures that are written in a 3GL language such as C or FORTRAN, linked into a shareable image, and registered in a database schema. External procedures are new in V7.0.

  External routines are available on all platforms.

  For more information, see Section 2.6.4 and the Create Routine Statement.

- Creating stored functions

  In addition to stored procedures, you can now define stored functions using the CREATE MODULE statement. A stored function is invoked by using the function name in a value expression.

  For more information, see the CREATE MODULE Statement, the Compound Statement, and the RETURN Control Statement.

- Returning the value of a stored function

  SQL provides the RETURN statement, which returns the result of a stored function.

  See the RETURN Control Statement for more information.

- DROP MODULE CASCADE and DROP MODULE RESTRICT implemented

  See the DROP MODULE Statement for more information.

- DROP PROCEDURE and DROP FUNCTION for external routines and stored routines implemented

  See the Drop Routine Statement for more information.

- CALL statement in a compound statement

  You can now use the CALL statement within a compound statement and, therefore, in a stored procedure or function to call another stored procedure.

  The CALL statement can also invoke external procedures.

  For more information, see the CALL Statement for Compound Statements.

- New SIGNAL statement

  SQL now adds a new SIGNAL statement for use within a compound statement.

  SIGNAL accepts a single character value expression that is used as the SQLSTATE. The current routine and all calling routines are terminated and the signaled SQLSTATE is passed to the application.

  For more information, see the SIGNAL Control Statement.

- Using the DEFAULT clause, CONSTANT clause, and UPDATABLE clause when declaring variables within compound statements

  Oracle Rdb includes full support in SQL for the CONSTANT, UPDATABLE, and DEFAULT clauses on declared variables within compound statements.

The default can be any value expression including subqueries, conditional, character, date/time, and numeric expressions. Additionally, Oracle Rdb can now inherit the default from the named domain if one exists.

The CONSTANT clause changes the variable into a declared constant that cannot be updated. If you use the CONSTANT clause, you must also have used the DEFAULT clause to ensure the variable has a value.

The UPDATABLE clause allows a variable to be updated through a SET assignment, an INTO assignment (as part of an INSERT, UPDATE, or SELECT statement), an equality (=) comparison, or as a parameter to a procedure OUT or INOUT parameter.

For more information, see the Compound Statement.

- Obtaining the connection name using the GET DIAGNOSTIC statement

  You can now obtain the current connection name in a variable or parameter from within a stored function, stored procedure, and a multistatement block using the GET DIAGNOSTICS statement.

  For more information, see the GET DIAGNOSTICS Statement.

- Support for the Shift_JIS character set

  Oracle Rdb includes support for the Shift_JIS character set; a mixed multi-octet character set.

  See Section 2.1 for more information.

- Altering RDB$SYSTEM storage area

  You can specify RDB$SYSTEM as the storage area name in the ALTER STORAGE AREA clause of an ALTER DATABASE statement. See ALTER DATABASE Statement for more information.

- Enhancements for the SQL SHOW statement

  The SQL SHOW statement displays the new features affecting data definition, stored routines, and external routines.

  For more information, see the SHOW Statement.

- The keyword ROWID

  You can use keyword ROWID as a synonym for the keyword DBKEY.

- COUNT function enhancements

  You can now specify:

  - COUNT (*)
  - COUNT (value-expr)

    - COUNT (DISTINCT value-expr)

  See Section 2.6.3.1 for more detail.

- Specifying the new dialect ORACLE LEVEL1

  You can now specify the ORACLE LEVEL1 dialect for the interactive SQL and dynamic SQL environments. This dialect is similar to the SQL92 dialect. For more information, see SET DIALECT Statement.

- Two new basic predicates added for inequality comparisons

  These new basic predicates are:

  ```
  ^=
  !=
  ```

  The != predicate is available only if you set your dialect to ORACLE LEVEL1. See Section 2.7.1 for more information on basic predicates.

- Enhancements to the NULL keyword

  The NULL keyword can be used as a value expression. For example, in a SELECT statement. See Section 2.6.1.

OpenVMS VAX OpenVMS Alpha

- Specifying C_PROTOTYPES=file-name

  The SQL module language C_PROTOTYPES qualifier now accepts a file name. See Section 3.5 for more information. ♦

Digital UNIX

- Editing in interactive SQL

  On Digital UNIX, you can use the EDIT statement within interactive SQL. It works similar to the SQL EDIT statement on OpenVMS. For more information, see the EDIT Statement. ♦

Digital UNIX

- Support for Pascal and FORTRAN on Oracle Rdb for Digital UNIX

  Oracle Rdb for Digital UNIX now supports the DEC FORTRAN and DEC Pascal languages for the SQL precompiler and the SQL module processor. ♦

- New command line qualifier for precompiled SQL

  Precompiled SQL now has the –[no]extend_source qualifier on the Digital UNIX platform.

  See Chapter 4 for more information.

# 6 SQL Statements

This chapter describes the syntax and semantics of all statements in SQL. SQL statements include data definition statements; data manipulation statements; statements that control the environment and program flow; and statements that give information.

See Chapter 2 in Volume 1 for detailed descriptions of the language and syntax elements referred to by the syntax diagrams in this chapter.

## ALTER DATABASE Statement

Alters a database in any of the following ways:

- For single-file and multifile databases, the ALTER DATABASE statement changes the characteristics of the database root file.

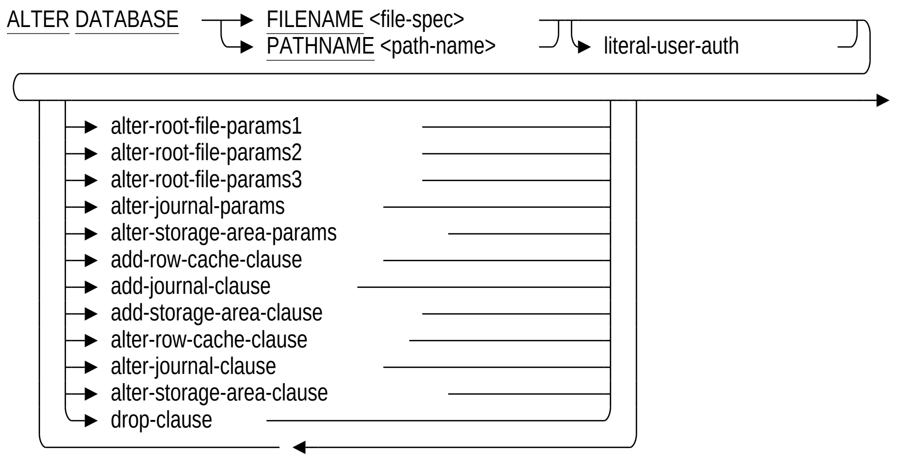
  The ALTER DATABASE statement lets you override certain characteristics specified in the database root file parameters of the CREATE DATABASE statement, such as whether or not a snapshot file is disabled. In addition, ALTER DATABASE lets you control other characteristics you cannot specify in the CREATE DATABASE database root file parameters, such as whether or not after-image journaling is enabled.
- For single-file and multifile databases, the ALTER DATABASE statement changes the storage area parameters.
- For multifile databases *only*, the ALTER DATABASE statement adds, alters, or deletes storage areas.

### Environment

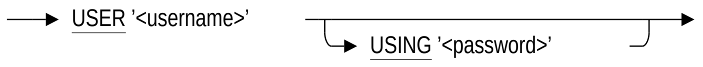
You can use the ALTER DATABASE statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

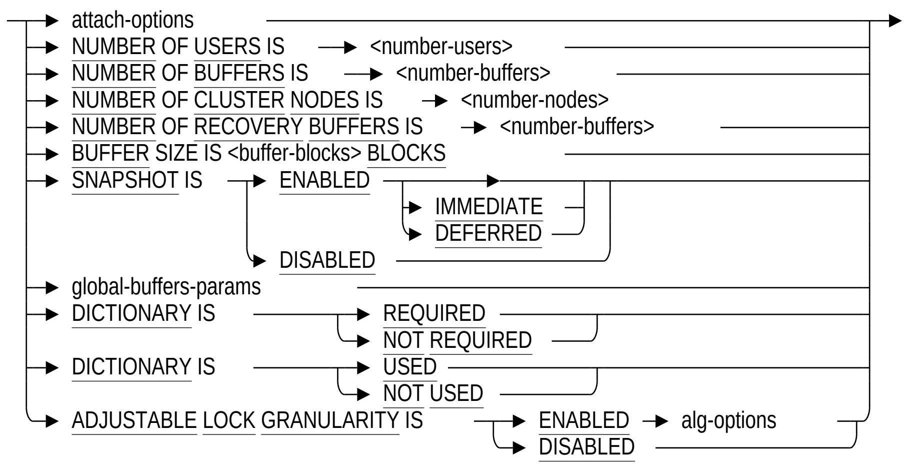
## Format



literal-user-auth =

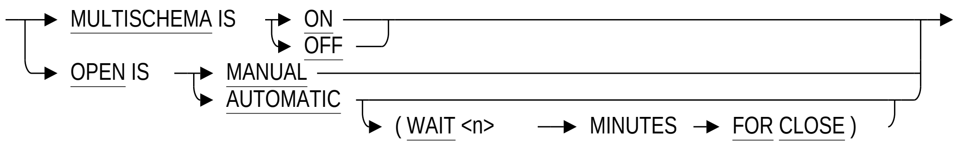


alter-root-file-params1 =

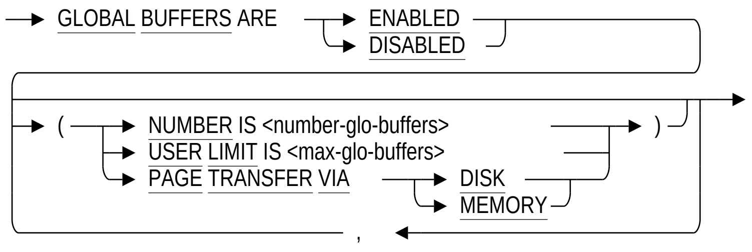


## ALTER DATABASE Statement

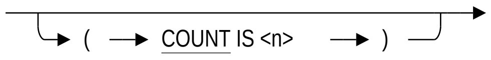
attach-options =



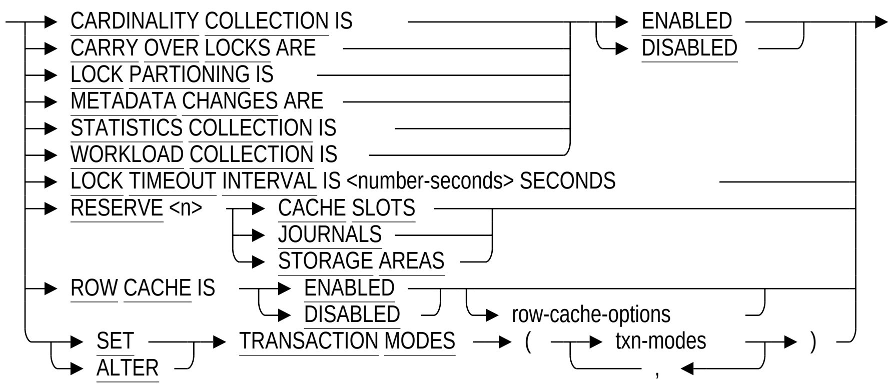
global-buffer-params=



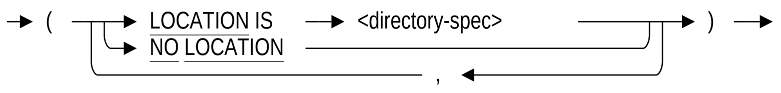
alg-options =



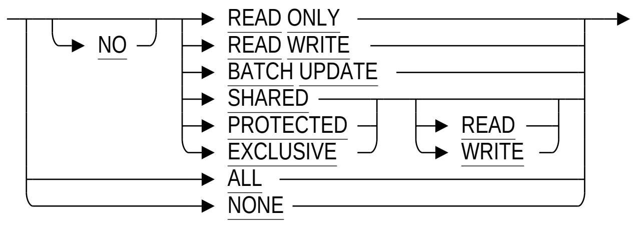
alter-root-file-params2 =



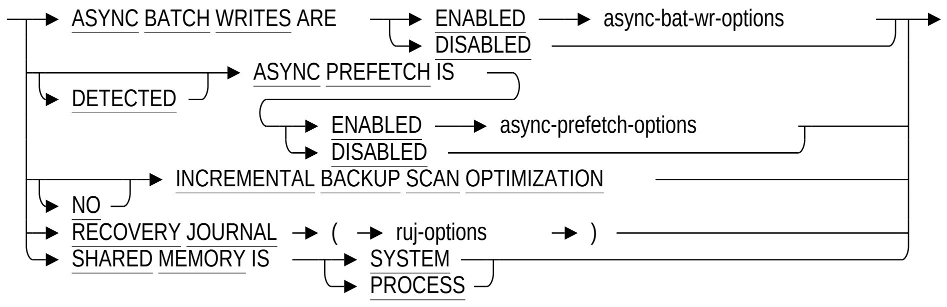
row-cache-options =



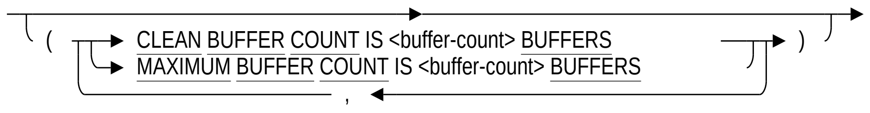
txn-modes =



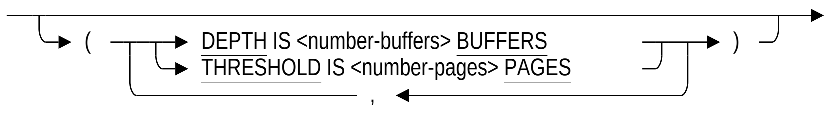
alter-root-file-params3 =



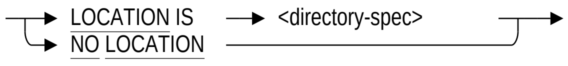
asynch-bat-wr-options =


(
CLEAN BUFFER COUNT IS <buffer-count> BUFFERS
MAXIMUM BUFFER COUNT IS <buffer-count> BUFFERS
,
)

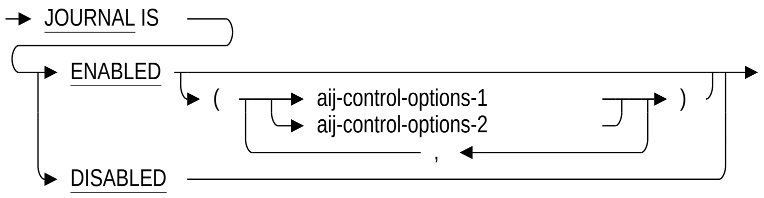
async-prefetch-options =


(
DEPTH IS <number-buffers> BUFFERS
THRESHOLD IS <number-pages> PAGES
,
)

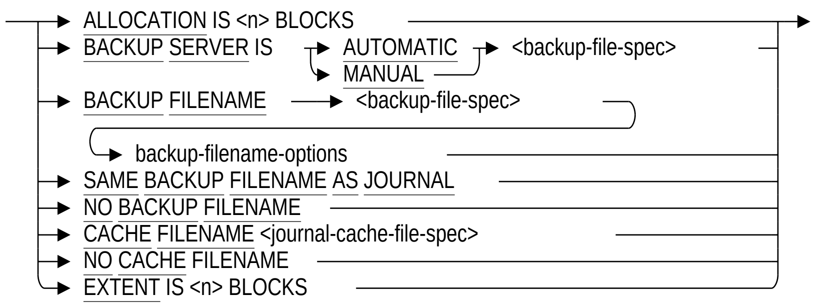
ruj-options =



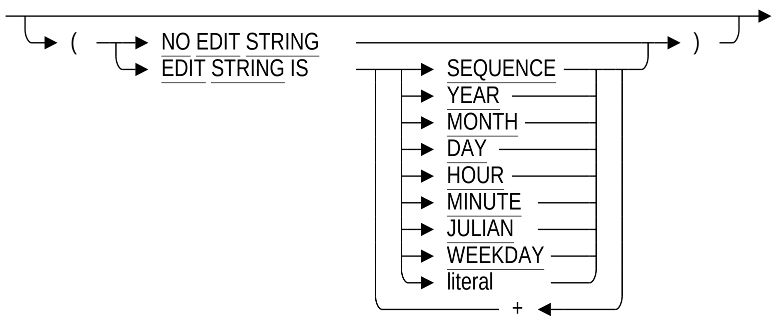
alter-journal-params =



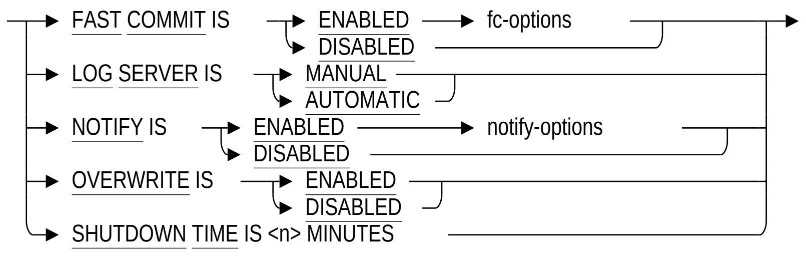
aij-control-options-1 =



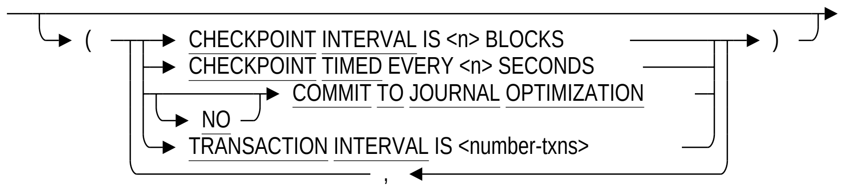
backup-filename-options =



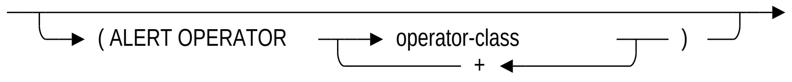
aij-control-options-2 =



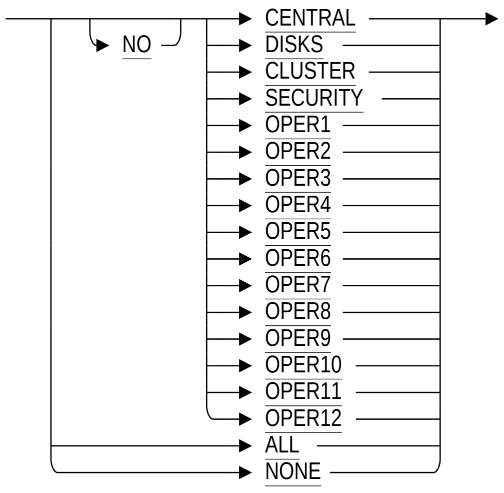
fc-options =


(
CHECKPOINT INTERVAL IS <n> BLOCKS
CHECKPOINT TIMED EVERY <n> SECONDS
COMMIT TO JOURNAL OPTIMIZATION
NO
TRANSACTION INTERVAL IS <number-txns>
,
)

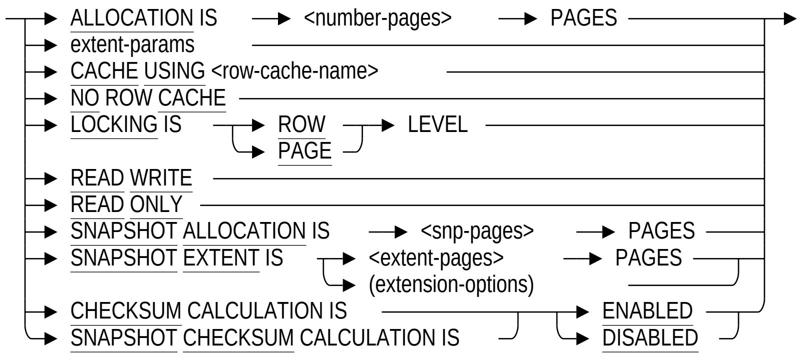
notify-options =



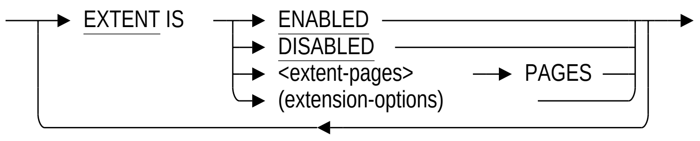
operator-class =



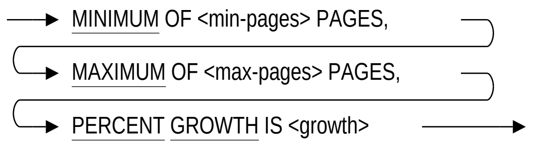
alter-storage-area-params =



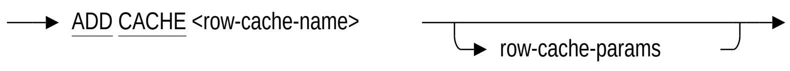
extent-params =



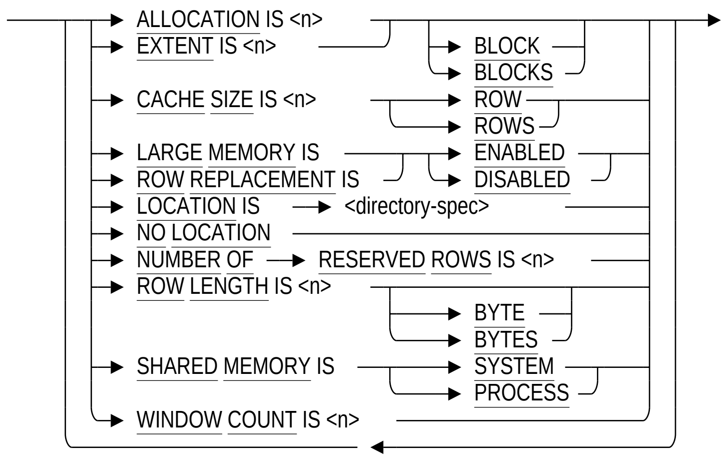
extension-options =


MINIMUM OF <min-pages> PAGES,
MAXIMUM OF <max-pages> PAGES,
PERCENT GROWTH IS <growth>

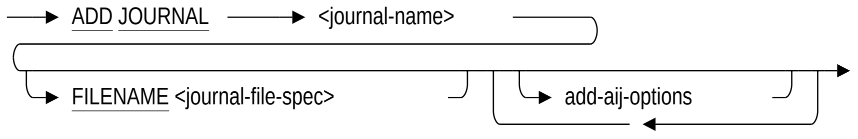
add-row-cache-clause =



row-cache-params =

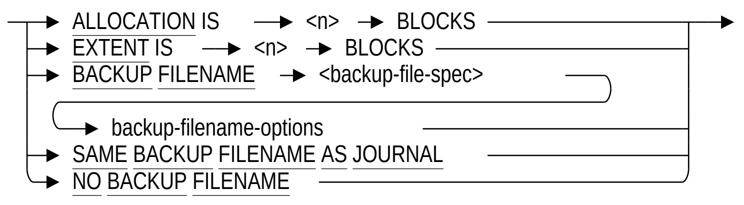


add-journal-clause =

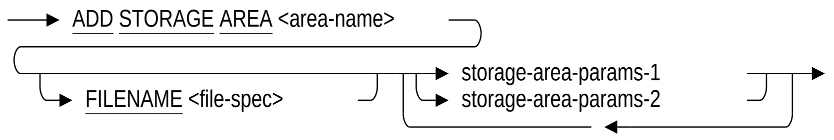


## ALTER DATABASE Statement

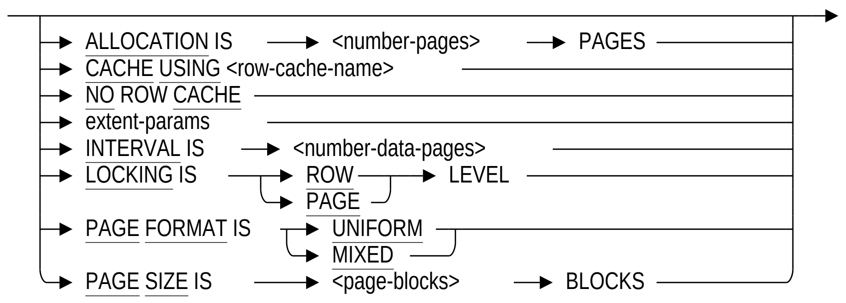
add-aij-options =



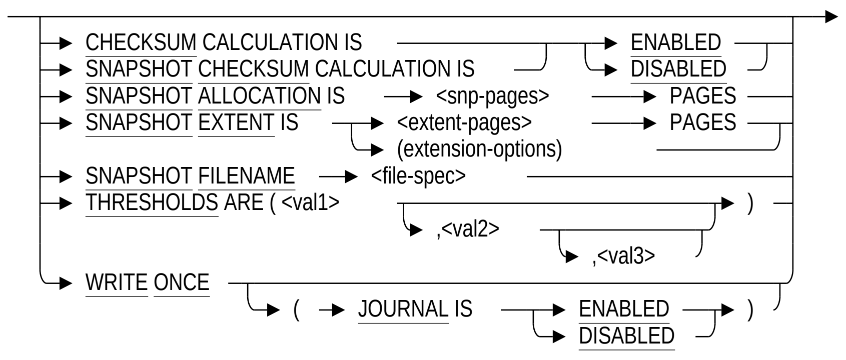
add-storage-area-clause =



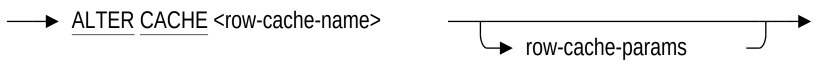
storage-area-params-1 =



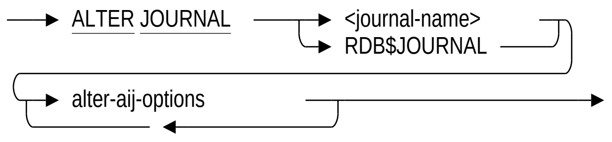
storage-area-params-2 =



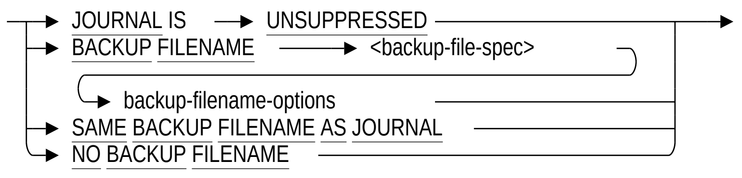
alter-row-cache-clause =



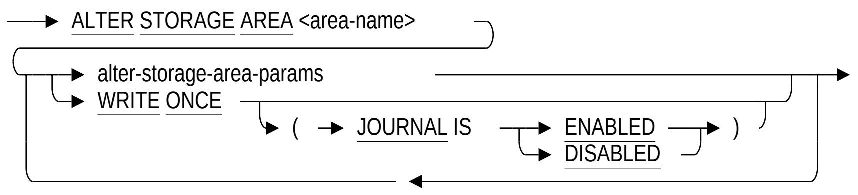
alter-journal-clause =



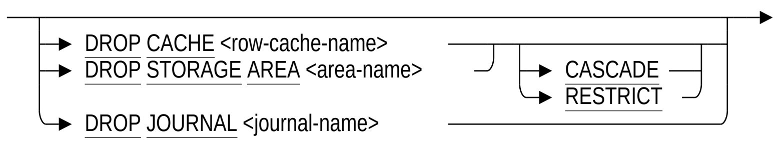
alter-aij-options =



alter-storage-area-clause =



drop-clause =


DROP CACHE <row-cache-name>
DROP STORAGE AREA <area-name>
CASCADE
RESTRICT
DROP JOURNAL <journal-name>

## Arguments

**FILENAME file-spec**
**PATHNAME path-name**
Identifies the database root file associated with the database. If you specify a repository path name, the path name indirectly specifies the database root file. The ALTER DATABASE statement does not change any definitions in the repository, so there is no difference in the effect of the PATHNAME and FILENAME arguments.

OpenVMS VAX OpenVMS Alpha

If you specify PATHNAME, SQL does not use the repository's fully qualified name. Instead, SQL uses the name stored as the user-supplied name in the repository. In the following example, SQL uses the name TEST as the file name, not DB$DISK:[DBDIR]TEST.RDB. As a result, the database root file must be located in your present working directory or the database name must be a logical name when you use the PATHNAME clause.

```
$ REPOSITORY OPERATOR
    .
    .
    .
CDO> show database/full test
Definition of database  TEST
|   database uses RDB database TEST
|   database in file TEST
|   |   fully qualified file DB$DISK:[DBDIR]TEST.RDB;
|   |   user-specified file DB$DISK:[DBDIR]test.rdb
```

If the database referred to in the PATHNAME or FILENAME argument has been attached, the ALTER DATABASE statement will fail with a file access conflict error.

The PATHNAME argument can only be specified on OpenVMS platforms. ♦

**literal-user-auth**
Specifies the user name and password for access to databases, particularly remote database.

This literal lets you explicitly provide user name and password information in the ALTER DATABASE statement.

**USER 'username'**
A character string literal that specifies the operating system user name that the database system uses for privilege checking. This clause also sets the value of the SYSTEM_USER value expression.

**USING 'password'**
A character string literal that specifies the user's password for the user name specified in the USER clause.

**alter-root-file-params1**
**alter-root-file-params2**
**alter-root-file-params3**
Parameters that control the characteristics of the database root file associated with the database or that control the characteristics that apply to the entire database. You can specify these parameters for either single-file or multifile databases except as noted in the individual parameter descriptions. For more information about database parameters and details about how they affect performance, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

The ALTER DATABASE statement does not let you change all database root file parameters that you can specify in the CREATE DATABASE statement. You must use the EXPORT and IMPORT statements to change a number of storage area parameters. For more information on changing storage area parameters, see the IMPORT Statement.

**MULTISCHEMA IS ON**
**MULTISCHEMA IS OFF**
Specifies the multischema attribute for the database. If a database has the multischema attribute, you can create multiple schemas in that database and group them within catalogs. The MULTISCHEMA IS ON option is the default for databases created with the multischema attribute. MULTISCHEMA IS OFF is the default for databases created without the multischema attribute.

You can create a database using the CREATE DATABASE MULTISCHEMA IS ON clause, but you cannot use ALTER DATABASE MULTISCHEMA IS OFF to take away the multischema attribute. Once a database has the multischema attribute, you cannot change it.

For more information about multischema databases, see Section 2.2.3.

**OPEN IS MANUAL**
**OPEN IS AUTOMATIC**
Specifies whether or not the database must be explicitly opened before users can attach to it. The default, OPEN IS AUTOMATIC, means that any user can open a previously unopened or a closed database by attaching to it and executing a statement. The OPEN IS MANUAL option means that a privileged user must issue an explicit OPEN statement through Oracle RMU, the Oracle Rdb management utility, before other users can attach to the database.

To issue the RMU Open command, you must have the RMU$OPEN privilege for the database.

The OPEN IS MANUAL option limits access to databases.

You will receive an error message if you specify both OPEN IS AUTOMATIC and OPEN IS MANUAL options.

**WAIT n MINUTES FOR CLOSE**
Specifies the amount of time that Oracle Rdb waits before automatically closing a database. If anyone attaches during that wait time, the database is not closed.

The default value for n is zero (0) if the WAIT clause is not specified. The value for n can range from zero (0) to 35,791,394. However, Oracle Rdb does not recommend using large values.

**NUMBER OF USERS IS number-users**
Limits the maximum number of users allowed to access the database at one time. Specify this clause only if the database named in the ALTER DATABASE statement refers to a multifile database.

The default is 50 users. After the maximum is reached, the next user who tries to invoke the database receives an error message and must wait. The maximum number of users you can specify is 16368 and the minimum is 1 user.

Note that *number of users* is defined as the number of active attachments to the database. Therefore, if a single process is running one program but that program performs 12 attach operations, the process is responsible for 12 active users.

If you use the ALTER DATABASE statement to change the current number of users, the change is not journaled. Therefore, back up your database before making such a change. See the Usage Notes for important information about changes that are not journaled.

**NUMBER OF BUFFERS IS number-buffers**
The number of buffers SQL allocates for each process using this database. Specify an unsigned integer with a value greater than or equal to 2 and less than or equal to 32,767. The default is 20 buffers.

**NUMBER OF CLUSTER NODES IS number-nodes**
Sets the upper limit on the maximum number of VMScluster nodes from which users can access the shared database. Specify this clause only if the database named in the ALTER DATABASE statement refers to a multifile database. The

default is 16 nodes. The range is 1 to 96 nodes. The actual maximum limit is the current VMScluster node limit set by your system administrator.

The NUMBER OF VAXCLUSTER NODES clause has been retained for backward compatibility.

**NUMBER OF RECOVERY BUFFERS IS number-buffers**
Specifies the number of buffers allocated to the automatic recovery process that Oracle Rdb initiates after a system or process failure. This recovery process uses the recovery-unit journal file (.ruj file extension).

You can specify any number greater than or equal to 2 and less than or equal to 32,767. The default value for the NUMBER OF RECOVERY BUFFERS parameter is 40. If you have a large, multifile database and you work on a system with a large amount of memory, specify a large number of buffers. The result is faster recovery time. However, make sure your buffer pool does not exceed the amount of memory you can allocate for the pool.

Use the NUMBER OF RECOVERY BUFFERS option to increase the number of buffers allocated to the recovery process.

```
SQL> ALTER DATABASE FILENAME personnel
cont>  NUMBER OF RECOVERY BUFFERS IS 150;
```

This option is used only if the NUMBER OF RECOVERY BUFFERS value is larger than the NUMBER OF BUFFERS value. For more information on allocating recovery buffers, see the *Oracle Rdb7 Guide to Database Maintenance*.

**BUFFER SIZE IS buffer-blocks BLOCKS**
Specifies the number of blocks SQL allocates per buffer. You need to specify an unsigned integer greater than zero. The default buffer size is 3 times the PAGE SIZE value (6 blocks for the default PAGE SIZE of 2).

The buffer size is a global parameter and the number of blocks per page (or buffer) is constrained to less than 64 blocks per page. The page size can vary by storage area for multifile databases, and the page size should be determined by the sizes of the records that will be stored in each storage area.

When choosing the number of blocks per buffer, choose a number so that a round number of pages fits in the buffer. In other words, the buffer size is wholly divisible by all page sizes for all storage areas in your multifile database. For example, if you have three storage areas with page sizes of 2, 3, and 4 blocks each respectively, choosing a buffer size of 12 blocks ensures optimal buffer utilization. In contrast, choosing a buffer size of 8 wastes 2 blocks per buffer for the storage area with a page size of 3 pages. Oracle Rdb reads as many pages as fit into the buffer; in this instance it reads two 3-block pages into the buffer, leaving 2 wasted blocks.

The altered buffer size must allow for existing page sizes. You cannot specify a buffer size smaller than the largest existing page size.

**SNAPSHOT IS ENABLED IMMEDIATE**
**SNAPSHOT IS ENABLED DEFERRED**
Specifies when read/write transactions write database changes to the snapshot file used by read-only transactions.

The ENABLED IMMEDIATE option is the default and causes read/write transactions to write copies of rows they modify to the snapshot file, regardless of whether or not a read-only transaction is active. Although ENABLED IMMEDIATE is the default, if you set snapshots ENABLED DEFERRED, you must specify both ENABLED and IMMEDIATE options to return the database to the default setting.

The ENABLED DEFERRED option lets read/write transactions avoid writing copies of rows they modify to the snapshot file (unless a read-only transaction is already active). Deferring snapshot writing in this manner improves the performance for the read/write transaction. However, read-only transactions that start after an active read/write transaction starts must wait for all active read/write users to complete their transactions.

**SNAPSHOT IS DISABLED**
Specifies that snapshot writing be disabled. Snapshot writing is enabled by default.

**GLOBAL BUFFERS ARE ENABLED**
**GLOBAL BUFFERS ARE DISABLED**
Specifies whether or not Oracle Rdb maintains one global buffer pool per VMScluster node for each database. By default, Oracle Rdb maintains a local buffer pool for each user (GLOBAL BUFFERS ARE DISABLED). For more than one user to use the same page, each must read it from the disk into their local buffer pool. A page in the global buffer pool can be read by more than one user at the same time, although only one user reads the page from the disk into the global buffer pool. Global buffers improve performance because the I/O is reduced, and memory is better utilized.

**Note**

In database parameter syntax, a “user” is defined as an attach to the database, not as a person who uses the database.

**NUMBER IS number-glo-buffers**
Specifies the total number of buffers in the global buffer pool. This number appears as "global buffer count" in RMU Dump command output. Base this value on the database users' needs and the number of attachments. The default is the maximum number of attachments multiplied by 5.

**Note**

Do not confuse the NUMBER IS parameter with the NUMBER OF BUFFERS IS parameter. The NUMBER OF BUFFERS IS parameter determines the default number of buffers Oracle Rdb allocates to each user's process that attaches to the database. The NUMBER OF BUFFERS IS parameter applies to, and has the same meaning for, local and global buffering. The NUMBER IS parameter has meaning only within the context of global buffering.

You can override the default number of user-allocated buffers by defining a value for the logical name RDM$BIND_BUFFERS or for the configuration parameter RDB_BIND_BUFFERS. For more information on user-allocated buffers, see *Oracle Rdb7 Guide to Database Performance and Tuning*.

Although you can change the NUMBER IS parameter on line, the change does not take effect until the next time the database is opened.

**USER LIMIT IS max-glo-buffers**
Specifies the maximum number of global buffers each user allocates. Because global buffer pools are shared by all users, you must define an upper limit on how many global buffers a single user can allocate. This limit prevents a user from defining RDM$BIND_BUFFERS or RDB_BIND_BUFFERS to use all the buffers in the global buffer pool. The user limit cannot be greater than the total number of global buffers. The default is 5 global buffers.

Decide the maximum number of global buffers a user can allocate by dividing the total number of global buffers by the total number of users for whom you want to guarantee access to the database. For example, if the total number of global buffers is 200 and you want to guarantee access to the database for at least 10 users, set the maximum number of global buffers per user to 20.

For maximum performance on a VMScluster system, tune the two global buffer parameters on each node in the cluster using the RMU Open command with the Global_Buffers qualifier.

Although you can change the USER LIMIT IS parameter on line, the change does not take effect until the next time the database is opened.

The NUMBER IS and USER LIMIT IS parameters are the only two buffer parameters specific to global buffers. They are in effect on a per node basis rather than a per process basis.

**PAGE TRANSFER VIA DISK**
**PAGE TRANSFER VIA MEMORY**
Specifies whether Oracle Rdb transfers (flushes) pages to disk or to memory.

When you specify PAGE TRANSFER VIA MEMORY, processes on a single node can share and update database pages in memory without transferring the pages to disk. It is not necessary for a process to write a modified page to disk before another process accesses the page.

The default is to DISK. If you specify PAGE TRANSFER VIA MEMORY, the database must have the following characteristics:

- The NUMBER OF CLUSTER NODES must equal one.
- GLOBAL BUFFERS must be enabled.
- After-image journaling must be enabled.
- FAST COMMIT must be enabled.

If the database does not have these characteristics, Oracle Rdb will perform page transfers via disk.

For more information about page transfers, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**DICTIONARY IS REQUIRED**
**DICTIONARY IS NOT REQUIRED**
OpenVMS VAX OpenVMS Alpha
Specifies whether or not definition statements issued for the database must also be stored in the repository. If you specify the REQUIRED option, any data definition statements issued after an ATTACH or DECLARE ALIAS statement that does not specify the PATHNAME argument fails.

If you specify the DICTIONARY argument in an ALTER DATABASE statement, you cannot specify any other database root file or storage area parameters.

If you omitted the PATHNAME clause from the database root file parameters in the CREATE DATABASE statement that created the database, SQL generates an error if you specify DICTIONARY IS REQUIRED in an ALTER DATABASE statement for the same database. This is not true if you use the INTEGRATE statement with the CREATE PATHNAME clause to copy database definitions to the repository before specifying the DICTIONARY IS REQUIRED clause in an ALTER DATABASE statement for that database.

This clause can be specified only on OpenVMS platforms. ♦

**DICTIONARY IS USED**
**DICTIONARY IS NOT USED**

OpenVMS VAX OpenVMS Alpha Specifies whether or not to remove the link between the repository and the database. If you specify the DICTIONARY IS NOT USED clause, the definitions in both the repository and database are still maintained. After removing the links, you can integrate the database to a new repository.

The DICTIONARY IS USED clause is the default. ♦

**ADJUSTABLE LOCK GRANULARITY IS ENABLED**
**ADJUSTABLE LOCK GRANULARITY IS DISABLED**

Enables or disables whether or not the database system automatically maintain as few locks as possible on database resources. The default, ENABLED, results in fewer locks against the database. However, if contention for database resources is high, the automatic adjustment of locks can become a CPU drain. You can trade more restrictive locking for less CPU usage in such databases by disabling adjustable lock granularity.

Always enable adjustable lock granularity if you are going to fetch more than 64,000 rows from a table in your database. If you disable adjustable lock granularity and attempt to fetch more than 64,000 rows, you receive an error message.

**COUNT IS n**

Specifies the number of levels on the page lock tree used to manage locks. For example, if you specify COUNT IS 3, the fanout factor is (10, 100, 1000). Oracle Rdb locks a range of 1000 pages and adjusts downward to 100 and then to 10 and then to 1 page when necessary.

If the COUNT IS clause is omitted, the default is 3. The value of n can range from 1 through 8.

**CARDINALITY COLLECTION IS ENABLED**
**CARDINALITY COLLECTION IS DISABLED**

Specifies whether or not the optimizer records cardinality updates in the system table. When enabled, the optimizer collects cardinalities for the table and non-unique indexes as rows are inserted or deleted from tables. The update of the cardinalities is performed at commit time, if sufficient changes have accumulated, or at disconnect time.

In high update environments, it may be more convenient to disable cardinality updates. If you disable this feature, you should manually maintain the cardinalities using the RMU Collect Optimizer_Statistics command so that the optimizer is given the most accurate values for estimation purposes.

Cardinality collection is enabled by default.

**CARRY OVER LOCKS ARE ENABLED**
**CARRY OVER LOCKS ARE DISABLED**
Enables or disables carry-over lock optimization. Carry-over lock optimization holds record and area locks across transactions. Carry-over locks are enabled by default and are available as an online database modification.

For more information on carry-over lock optimization, see the CREATE DATABASE Statement.

**LOCK PARTITIONING IS ENABLED**
**LOCK PARTITIONING IS DISABLED**
OpenVMS Alpha
Specifies whether more than one lock tree is used for the database or all lock trees for a database are mastered by one database resource tree.

When partitioned lock trees are enabled for a database, locks for storage areas are separated from the database resource tree and all locks for each storage area are independently mastered on the VMScluster node that has the highest traffic for that resource. OpenVMS determines the node that is using each resource the most and moves the resource hierarchy to that node.

You cannot enable lock partitioning for single-file databases. You should not enable lock partitioning for single-node systems, because all lock requests are local on single-node systems.

By default, lock partitioning is disabled.

This clause is for the OpenVMS Alpha platform only. ♦

**METADATA CHANGES ARE ENABLED**
**METADATA CHANGES ARE DISABLED**
Specifies whether or not data definition changes are allowed to the database. This attribute becomes effective at the next database attach and affects all ALTER, CREATE, and DROP statements (except ALTER DATABASE which is needed for database tuning) and the GRANT, REVOKE, and TRUNCATE TABLE statements. For example:

```
SQL> CREATE DATABASE FILENAME sample;
SQL> CREATE TABLE t (a INTEGER);
SQL> DISCONNECT ALL;
SQL> ALTER DATABASE FILENAME sample
cont> METADATA CHANGES ARE DISABLED;
SQL> ATTACH 'FILENAME sample';
SQL> CREATE TABLE s (b INTEGER);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-E-NOMETADATA, metadata operations are disabled
```

The METADATA CHANGES ARE DISABLED clause prevents data definition changes to the database.

The METADATA CHANGES ARE ENABLED clause allows data definition changes to the database by users granted the DBADMIN privilege.

METADATA CHANGES ARE ENABLED is the default.

**STATISTICS COLLECTION IS ENABLED**
**STATISTICS COLLECTION IS DISABLED**
Specifies whether the collection of statistics for the database is enabled or disabled. When you disable statistics for the database, statistics are not displayed for any of the processes attached to the database. Statistics are displayed using the RMU Show Statistics command.

The default is STATISTICS COLLECTION IS ENABLED. You can disable statistics using the ALTER DATABASE and IMPORT statements.

For more information on the RMU Show Statistics command, see the *Oracle RMU Reference Manual*.

You can enable statistics collection by defining the logical name RDM$BIND_STATS_ENABLED or the configuration parameter RDB_BIND_STATS_ENABLED. For more information about when to use statistics collection, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**WORKLOAD COLLECTION IS ENABLED**
**WORKLOAD COLLECTION IS DISABLED**
Specifies whether or not the optimizer records workload information in the system table RDB$WORKLOAD. The WORKLOAD COLLECTION IS ENABLED clause creates this system table if it does not exist. If you later disable workload collection, the RDB$WORKLOAD system table is not deleted, nor is the data deleted.

A workload profile is a description of the interesting table and column references used by queries in a database work load. When workload collection is enabled, the optimizer collects and records these references in the RDB$WORKLOAD system table. This work load is then processed by the RMU Collect Statistics command which records useful statistics about the work load. These workload statistics are used by the optimizer at run time to deliver more accurate access strategies.

Workload collection is disabled by default.

**LOCK TIMEOUT INTERVAL IS number-seconds SECONDS**
Specifies the number of seconds for processes to wait during a lock conflict before timing out. The number can be between 1 and 65,000 seconds.

Specifying 0 is interpreted as no lock timeout interval being set. It is not interpreted as 0 seconds.

The lock timeout interval is database-wide; it is used as the default and the upper limit when determining the timeout interval. For example, if the database definer specified LOCK TIMEOUT INTERVAL IS 25 SECONDS in the ALTER DATABASE statement, and a user of that database specified SET TRANSACTION WAIT 30 or changed the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL or configuration parameter RDB_BIND_LOCK_TIMEOUT_INTERVAL to 30, SQL still uses the interval 25. For more information on timeout intervals, see the *Oracle Rdb7 Guide to Distributed Transactions*.

**RESERVE n CACHE SLOTS**
Specifies the number of row cache areas for which slots are reserved in the database. If your database is a single file database, you have only one cache slot and cannot reserve additional slots.

You can use the RESERVE CACHE SLOTS clause to reserve slots in the database root file for future use by the ADD CACHE clause. Row cache areas can be added only if there are row cache slots available. Slots become available after a DROP CACHE clause or a RESERVE CACHE SLOTS clause.

The number of reserved slots for row cache areas cannot be decreased once the RESERVE clause is issued. If you reserve 10 slots and later reserve 5 slots, you have a total of 15 reserved slots for row cache areas.

Reserving row cache slots is an offline operation (requiring exclusive database access).

**RESERVE n JOURNALS**
Specifies the number of journal files for which to reserve slots in the database. The number of slots for journal files must be a positive number greater than zero.

This feature is additive in nature. In other words, the number of reserved slots for journal files cannot be decreased once the RESERVE clause has been issued. If you reserve 10 slots and later reserve 5 slots, you have a total of 15 reserved slots for journal files plus 1 slot (totaling 16 reserved slots) because you initially get 1 pre-reserved slot.

You must reserve slots or delete an existing journal file before you can add new journal files to the database.

You cannot reserve journal files for a single-file database.

**RESERVE n STORAGE AREAS**

Specifies the number of storage areas for which to reserve slots in the database. The number of slots for storage areas must be a positive number greater than zero.

You can use the RESERVE STORAGE AREA clause to reserve slots in the database root file for future use by the ADD STORAGE AREA clause of the ALTER DATABASE statement. Storage areas can be added only if there are storage area slots available. Slots become available after a DROP STORAGE AREA clause or a RESERVE STORAGE AREA clause is issued.

This feature is additive in nature. In other words, the number of reserved slots for storage areas cannot be decreased once the RESERVE clause is issued. If you reserve 10 slots and later reserve 5 slots, you have a total of 15 reserved slots for storage areas.

You must reserve slots or delete an existing storage area before you can add new storage areas to the database.

If you do not specify the RESERVE STORAGE AREA clause, the default number of reserved storage areas is zero.

**ROW CACHE IS ENABLED**
**ROW CACHE IS DISABLED**

Specifies whether or not you want Oracle Rdb to enable the row caching feature.

Enabling cache support does not affect database operations until a cache is created and assigned to one or more storage areas.

When the row caching feature is disabled, all previously created and assigned cache areas remain in existence for future use when the row caching feature is enabled.

**LOCATION IS directory-spec**

Specifies the name of the backing store directory to which row cache information is written. The database system generates a file name (row-cache-name.rdc) automatically for each row cache area at checkpoint time. Specify a device name and directory name only, enclosed within by single quotation marks. The file name is the row-cache-name specified when creating the row cache area. By default, the location is the directory of the database root file. These .rdc files are permanent database backing store files.

The LOCATION clause of the CREATE CACHE, ADD CACHE, or ALTER CACHE clause overrides this location which is the default for the database.

**NO LOCATION**
Removes the location previously specified in a LOCATION IS clause for the database for the row cache area. If you specify NO LOCATION, the row cache location becomes the directory of the database root file.

**SET TRANSACTION MODES**
Enables only the modes specified, disabling all other previously defined modes. This is an offline operation and requires exclusive database access. For example, if a database is used for read-only access and you want to disable all other transaction modes, specify the following statement:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>   SET TRANSACTION MODES (READ ONLY);
```

Specifying a negated txn-mode or specifying NONE disables all transaction usage. Disabling all transaction usage would be useful when, for example, you want to perform major restructuring of the physical database. Execute the ALTER DATABASE statement to re-enable transaction modes or use Oracle RMU, the Oracle Rdb management utility.

**ALTER TRANSACTION MODES**
Enables or disables the modes specified leaving the previously defined or default modes enabled. This is an offline operation and requires exclusive database access.

If the current transaction modes are SHARED and READ ONLY and you want to add the EXCLUSIVE mode, use the following statement:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>   ALTER TRANSACTION MODES (EXCLUSIVE);
```

**txn-modes**
Specifies the transaction modes for the database.

| Mode | Description |
|---|---|
| | **Transaction Types** |
| [NO]READ ONLY | Allows read-only transactions on the database. |
| [NO]READ WRITE | Allows read/write transactions on the database. |

| Mode | Description |
|---|---|
| | **Transaction Types** |
| [NO] BATCH UPDATE | Allows batch-update transactions on the database. This mode executes without the overhead, or security, or a recovery-unit journal file. The batch-update transaction is intended for the initial loading of a database. Oracle Rdb recommends that this mode be disabled. |
| | **Reserving Modes** |
| [NO] SHARED [READ \| WRITE] | Allows other users to work with the specified tables. |
| [NO] PROTECTED [READ \| WRITE] | Allows other users to read the specified tables. |
| [NO] EXCLUSIVE [READ \| WRITE] | Allows no access to the specified tables. |
| ALL | Allows other users to work with the specified tables. |
| NONE | Allows no access to the specified tables. |

For detailed information about the txn-modes, see the SET TRANSACTION Statement.

**ASYNC BATCH WRITES ARE ENABLED**
**ASYNC BATCH WRITES ARE DISABLED**
Specifies whether asynchronous batch-writes are enabled or disabled.

Asynchronous batch-writes allow a process to write batches of modified data pages to disk asynchronously (the process does not stall while waiting for the batch-write operation to complete). Asynchronous batch-writes improve the performance of update applications without the loss of data integrity.

By default, batch-writes are enabled.

For more information about when to use asynchronous batch-writes, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

You can enable asynchronous batch-writes by defining the logical name RDM$BIND_ABW_ENABLED or the configuration parameter RDB_BIND_ABW_ENABLED.

**CLEAN BUFFER COUNT IS buffer-count**
Specifies the number of buffers to be kept available for immediate reuse.

The default is five buffers. The minimum value is 1; the maximum value can be as large as the buffer pool size.

You can override the number of clean buffers by defining the logical name RDM$BIND_CLEAN_BUF_CNT or the configuration parameter RDB_BIND_CLEAN_BUF_CNT. For information about how to set the values, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**MAXIMUM BUFFER COUNT IS buffer-count**
Specifies the number of buffers a process will write asynchronously.

The default is one-fifth of the buffer pool, but not more than 10 buffers. The minimum value is 2 buffers; the maximum value can be as large as the buffer pool.

You can override the number of buffers to be written asynchronously by defining the logical name RDM$BIND_BATCH_MAX or the configuration parameter RDB_BIND_BATCH_MAX. For information about how to set the values, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**ASYNC PREFETCH IS ENABLED**
**ASYNC PREFETCH IS DISABLED**
Specifies whether or not Oracle Rdb reduces the amount of time that a process waits for pages to be read from disk by fetching pages before a process actually requests the pages.

Prefetch can significantly improve performance, but it may cause excessive resource usage if it is used inappropriately. Asynchronous prefetch is enabled by default. For more information about asynchronous prefetch, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

You can enable asynchronous prefetch by defining the logical name RDM$BIND_APF_ENABLED or the configuration parameter RDB_BIND_APF_ENABLED.

**DEPTH IS number-buffers BUFFERS**
Specifies the number of buffers to prefetch for a process.

The default is one-quarter of the buffer pool, but not more than eight buffers. You can override the number of buffers specified in the CREATE or ALTER DATABASE statements by using the logical name RDM$BIND_APF_DEPTH or the configuration parameter RDB_BIND_APF_DEPTH.

You can also specify this option with the DETECTED ASYNC PREFETCH clause.

**DETECTED ASYNC PREFETCH IS ENABLED**
**DETECTED ASYNC PREFETCH IS DISABLED**
Specifies whether or not Oracle Rdb reduces the amount of time that a process waits for pages to be read from disk.

By using heuristics, detected asynchronous prefetch determines if an I/O pattern is sequential in behavior even if sequential I/O is not actually executing at the time. For example, when a LIST OF BYTE VARYING column is fetch, the heuristics detect that the pages being fetched are sequential and, therefore, fetch ahead asynchronously to avoid wait times when the page is really needed.

Detected asynchronous prefetch is enabled by default.

**THRESHOLD IS number-pages PAGES**
Specifies the number of pages to prefetch for a process. The default is one-quarter of the buffer pool, but not more than eight pages.

If you specify the THRESHOLD option, you must have also specified the DETECTED ASYNC PREFETCH clause. You receive an error if you attempt to specify the THRESHOLD option with the ASYNC PREFETCH clause.

**INCREMENTAL BACKUP SCAN OPTIMIZATION**
**NO INCREMENTAL BACKUP SCAN OPTIMIZATION**
Specifies whether Oracle Rdb checks each area's SPAM pages or each database page to find changes during incremental backup.

If you specify INCREMENTAL BACKUP SCAN OPTIMIZATION, Oracle Rdb checks each area's SPAM pages and scans the SPAM interval of pages only if the SPAM transaction number (TSN) is higher than the last full backup TSN, which indicates that a page in the SPAM interval has been updated since the last full backup operation.

Specify INCREMENTAL BACKUP SCAN OPTIMIZATION if your database has large SPAM intervals or infrequently occurring updates, and you want to increase the speed of incremental backups. If you disable the attribute (using the NO INCREMENTAL BACKUP SCAN OPTIMIZATION clause), you cannot enable it until *immediately* after the next full backup.

If you specify NO INCREMENTAL BACKUP SCAN OPTIMIZATION, Oracle Rdb checks each page to find changes during incremental backup.

Specify the NO INCREMENTAL BACKUP SCAN OPTIMIZATION clause if your database has frequently occurring updates, uses bulk-load operations, or does not use incremental backups, or if you want to improve run-time performance.

The default is INCREMENTAL BACKUP SCAN OPTIMIZATION.

**RECOVERY JOURNAL (LOCATION IS directory-spec)**
Specifies the location in which the recovery-unit journal (.ruj) file is written. Do not include node names, file names, or process concealed logical names in the directory-spec. Single quotation marks are required around the directory-spec. This clause overrides the RDMS$RUJ logical name or the RDB_RUJ configuration parameter.

If this clause is omitted, the default directory location is either:

- On OpenVMS, the device:[RDM$RUJ] or the location defined by the RDMS$RUJ logical name
- On Digital UNIX, the database rootfile directory ( . . . /database.rdb /database.ruj) or the location defined by the RDB_RUJ configuration parameter

See the *Oracle Rdb7 Guide to Database Maintenance* for more information on recovery-unit journal files.

OpenVMS VAX OpenVMS Alpha

Following is an example using this clause on an OpenVMS system:

```
SQL> ALTER DATABASE FILENAME SAMPLE
cont> RECOVERY JOURNAL (LOCATION IS 'SQL_USER1:[DBDIR.RECOVER]');♦
```

Digital UNIX

Following is an example using this clause on a Digital UNIX system:

```
SQL> ALTER DATABASE FILENAME sample
cont> RECOVERY JOURNAL (LOCATION IS '/tmp/dbdir');♦
```

**RECOVERY JOURNAL (NO LOCATION)**
Removes a location previously defined by a RECOVERY JOURNAL (LOCATION . . . ) clause or the location defined by the RDMS$RUJ logical name or the RDB_RUJ configuration parameter.

If you specify NO LOCATION, the recovery-unit journal file reverts to the the default directory location device:[RDM$RUJ] on OpenVMS or to the database rootfile directory ( . . . /database.rdb/database.ruj) on Digital UNIX. See the *Oracle Rdb7 Guide to Database Maintenance* for more information on recovery-unit journal files.

**SHARED MEMORY IS SYSTEM**
**SHARED MEMORY IS PROCESS**

OpenVMS Alpha

Determines whether database root global sections (including global buffers when enabled) are created in system space or process space. The default is PROCESS.

When you use global sections created in the process space, you and other users share physical memory and the OpenVMS operating system maps a row cache area to a private address space for each user. As a result, all users are limited by the free virtual address range and each use a percentage of memory in overhead. If many users are accessing the database, the overhead can be high. ♦

**JOURNAL IS ENABLED**
**JOURNAL IS DISABLED**

Specifies whether or not journaling is enabled.

If journal files already exist, the JOURNAL IS ENABLED clause simply restarts the journaling feature.

If no journal files exist when the ALTER DATABASE . . . JOURNAL IS ENABLED statement completes, an exception is raised. For example:

```
SQL> ALTER DATABASE FILENAME sample
cont> JOURNAL IS ENABLED;
%RDMS-F-NOAIJENB, cannot enable after-image journaling without any AIJ journals
```

Use the ADD JOURNAL clause to create journal files.

The ENABLED option can be followed by a list of database journal options.

All journal files remain unchanged but become inaccessible when you disable them. You cannot specify database journal options with the DISABLED option.

**ALLOCATION IS n BLOCKS**

Specifies the number of blocks allocated for the .aij file. The default and minimum is 512 blocks. Even if you specify a value less than 512 blocks, the .aij file is allocated 512 blocks.

For information on determining the allocation value, see the *Oracle Rdb7 Guide to Database Design and Definition*.

**BACKUP SERVER IS AUTOMATIC backup-file-spec**
**BACKUP SERVER IS MANUAL backup-file-spec**

Specifies whether the backup server runs automatically or manually.

If BACKUP SERVER IS MANUAL is specified, you must execute the RMU Backup After_Journal command manually. If BACKUP SERVER IS AUTOMATIC is specified, a special backup server runs when a journal file in the set is full and causes a switch over to another journal file.

The default is MANUAL.

**BACKUP FILENAME backup-file-spec**
Specifies the default file specification to be used by the backup server.

During execution, the backup server and the RMU Backup After_Journal command use this file specification as the name of the backup file. You can override this value by specifying a file name for the journal file using the RMU Backup After_Journal command.

**backup-filename-options**
Specifies whether or not the backup file name includes an edit string. When the EDIT STRING clause is used, the specified backup file name is edited by appending any or all of the edit string options listed in the following table.

| Edit String Option | Meaning |
|---|---|
| SEQUENCE | The journal sequence number of the first journal file in the backup operation. |
| YEAR | The current year expressed as a 4-digit integer. |
| MONTH | The current month expressed as a 2-digit integer (01-12). |
| DAY | The current day of the month expressed as a 2-digit integer (00-31). |
| HOUR | The current hour of the day expressed as a 2-digit integer (00-23). |
| MINUTE | The current minute of the hour expressed as a 2-digit integer (00-59). |
| JULIAN | The current day of the year expressed as a 3-digit integer (001-366). |
| WEEKDAY | The current day of the week expressed as a 1-digit integer (1-7) where 1 is Sunday and 7 is Saturday. |
| literal | Any string literal. This string literal is copied to the file specification. See Section 2.4.2.1 for more information about string literals. |

Use a plus sign (+) between multiple edit string options. The edit string should be 32 characters or less in length.

The default is NO EDIT STRING which means the BACKUP FILENAME supplied is all that is used to name the backup file.

**SAME BACKUP FILENAME AS JOURNAL**
During execution, the backup server assigns the same name to the backup file as it does to the journal file. This is a quick form of backup as a new file is created.

**Note**

Oracle Rdb recommends that you save the old journal file on tape or other media to prevent accidental purging of these files.

**NO BACKUP FILENAME**
Removes a previously established backup file specification.

**CACHE FILENAME journal-cache-file-spec**
The journal cache is a special file located on fast media, such as an ESE50, and only requires 65 blocks per node.

The electronic AIJ cache (ACE) device should be a fast medium; for example, a solid-state disk.

**NO CACHE FILENAME**
Removes a previously established cache file specification and disables the journal cache feature.

**EXTENT IS n BLOCKS**
Specifies the number of blocks of each .aij file extent. The default and minimum extent for .aij files is 512 blocks.

**FAST COMMIT IS ENABLED**
**FAST COMMIT IS DISABLED**
By default, Oracle Rdb writes updated database pages to the disk each time a transaction executes the COMMIT statement. If a transaction fails before committing, Oracle Rdb only needs to roll back (undo) the current failed transaction; it never has to redo previous successful transactions.

You can change the commit processing method by enabling journal fast commit processing. With journal fast commit enabled, Oracle Rdb keeps updated pages in the buffer pool (in memory) and does not write the pages to the disk when a transaction commits. The updated pages remain in the buffer pool until the process meets a condition specified by the database administrator or applications programmer. At the moment the condition is met (the checkpoint), all the pages the process updated for multiple transactions are written to the disk.

You can set a checkpoint for your process when:

- A fixed number of transactions are committed or aborted. You set this by defining the logical name RDM$BIND_CKPT_TRANS_INTERVAL or the configuration parameter RDB_BIND_CKPT_TRANS_INTERVAL.
- A specified time interval elapsed. You set this by specifying the CHECKPOINT TIMED EVERY *n* SECONDS clause.
- The after-image journal (.aij) file increased to a specified block size. You set this by specifying the CHECKPOINT INTERVAL IS *n* BLOCKS clause.

If a transaction fails, Oracle Rdb must undo the current, failed transaction and redo all the committed transactions since the last checkpoint. Redoing updates involves reading the .aij file and reapplying the changes to the relevant data pages.

The checkpoint interval value is set by the database administrator and applies to all processes attached to a database. Users can implement an alternate, process-specific method of checkpointing by defining the logical name RDM$BIND_CKPT_TRANS_INTERVAL or configuration parameter RDB_BIND_CKPT_TRANS_INTERVAL. The logical name or configuration parameter uses transaction count as the checkpoint. When fast commit processing is disabled, the logical name or configuration parameter is ignored. For more information about the RDM$BIND_CKPT_TRANS_INTERVAL logical name or the RDB_BIND_CKPT_TRANS_INTERVAL configuration parameter, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

Fast commit processing applies only to data updates: erase, modify, and store operations. Transactions that include data definition statements, such as create logical area or create index operations, force a checkpoint at the end of the transaction. If you do not specify values with the FAST COMMIT clause, the default values are applied.

**Note**

To enable FAST COMMIT, you must first enable after-image journaling.

### CHECKPOINT INTERVAL IS n BLOCKS

You can limit how many transactions the database recovery process (DBR) must redo by setting a checkpoint interval. Setting a checkpoint interval instructs Oracle Rdb to periodically transfer updated pages. This shortens recovery time.

The value you assign to the checkpoint interval specifies the number of blocks the .aij file is allowed to increase to before updated pages are transferred. For example, if you set the checkpoint interval value equal to 100, all processes transfer updated pages to the disk when 100 blocks were written to the .aij file since the last checkpoint. Thus all processes contribute to .aij growth.

If no checkpoint interval is established and a process completes 1000 transactions but fails during number 1001, the DBR must redo transactions 1 through 1000 and undo number 1001.

When a process attaches to the database, it writes a checkpoint record to the .aij file and notes the virtual block number (VBN) of the .aij file at which the checkpoint record is located. If the checkpoint is located at VBN 120 and the checkpoint interval is 100 blocks, the process checkpoints again when VBN 220 is reached.

A process will never checkpoint in the middle of a transaction. Because all processes contribute to .aij file growth, a process may be able to commit many transactions before checkpointing if update activity by other processes is low. Conversely, if a process' first transaction is long and if update activity by other processes is high, the process may be forced to checkpoint when it commits its first transaction.

When the database checkpoint interval value is reached, Oracle Rdb executes the following steps:

1. Writes updated pages to the disk.
2. Writes a checkpoint record to the .aij file.
3. Updates the run-time user process block (RTUPB) for each process to indicate where the checkpoint record is stored in the .aij file.

   The RTUPB is a data structure in the database root file that maintains information on each process accessing the database. The database recovery process (DBR) uses the RTUPB checkpoint entry to determine where in the .aij file recovery must start.

**CHECKPOINT TIMED EVERY n SECONDS**
Assigns a value to the checkpoint interval specifying the number of seconds that can pass before updated pages are written. When the specified number of seconds elapsed, Oracle Rdb executes the checkpoint steps described in the previous section.

For example, if you specify TIMED EVERY 100 SECONDS, each process checkpoints when it completes a transaction after at least 100 seconds have passed since its last checkpoint.

You can set both a checkpoint based on time and a checkpoint based on .aij file growth; Oracle Rdb performs a checkpoint operation at whichever checkpoint it reaches first.

The following statement enables fast commit processing and specifies checkpoint intervals of 512 blocks and 12 seconds:

```
SQL> ALTER DATABASE FILENAME test1
cont> JOURNAL IS ENABLED
cont>    (FAST COMMIT ENABLED
cont>       (CHECKPOINT INTERVAL IS 512 BLOCKS,
cont>        CHECKPOINT TIMED EVERY 12 SECONDS)
cont>    );
```

**COMMIT TO JOURNAL OPTIMIZATION**
**NO COMMIT TO JOURNAL OPTIMIZATION**
If you enable COMMIT TO JOURNAL OPTIMIZATION when you enable fast commit processing, Oracle Rdb does not write commit information to the database root file. This option enhances performance in database environments that are update-intensive. Because of the prerequisites for enabling the journal optimization option, general-use databases or databases that have many read-only transactions may not benefit from this feature. For more information, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**Note**

If you specify COMMIT TO JOURNAL OPTIMIZATION, you must disable or defer snapshots.

If you change snapshots to ENABLED IMMEDIATE, then you must specify NO COMMIT TO JOURNAL OPTIMIZATION.

**TRANSACTION INTERVAL IS number-txns**
The TRANSACTION INTERVAL IS clause specifies the size of the transaction sequence number (TSN) range where *number-txns* equals the number of TSNs. Oracle Rdb uses transaction sequence numbers to ensure database integrity. When you specify NO COMMIT TO JOURNAL OPTIMIZATION, Oracle Rdb assigns TSNs to users one at a time. When you enable the journal optimization option, Oracle Rdb preassigns a range of TSNs to each user. Assigning a range of TSNs means that commit information need not be written to the database root for each transaction. Oracle Rdb writes all transaction information to the .aij file except for each user's allocated TSN range, which it writes to the root file.

The transaction interval value (the TSN range) must be a number between 8 and 1024. The default value is 256.

In general, if your database has few users or if all user sessions are long, select a large transaction interval. If your database has many users or if user sessions are short, select a smaller transaction interval.

**LOG SERVER IS MANUAL**
**LOG SERVER IS AUTOMATIC**
Specifies if the log server is activated manually or automatically. The default is manual.

Multiple-user databases with medium to high update activity can experience after-image journal (.aij) file bottlenecks. To alleviate these bottlenecks, you can specify the LOG SERVER clause to transfer log data to the .aij file either automatically or manually. On a single node with ALS, there is no AIJ locking.

If the log server is set to MANUAL, you must execute the RMU Server After_Journal command with the Start qualifier to start the log server. In this case, the database must already be open. If the OPEN IS MANUAL clause was specified, an explicit RMU Open command needs to be executed before the log server is started. If the OPEN IS AUTOMATIC clause was specified, at least one user should be attached to the database before the log server is started.

If the log server is set to AUTOMATIC, the log server starts when the database is opened, automatically or manually, and is shut down when the database is closed.

For more information on setting log servers, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**NOTIFY IS ENABLED**
**NOTIFY IS DISABLED**

OpenVMS VAX OpenVMS Alpha

Specifies whether system notification is enabled or disabled.

When the system notification is enabled, the system is notified in the event of a catastrophic journaling event such as running out of disk space. For example, if the NOTIFY and OVERWRITE options are enabled and a modified after-image journal file is written over, the database is no longer recoverable and a message is sent to the system.

If you specify the NOTIFY IS ENABLED clause and do not specify the ALERT OPERATOR clause, the operator classes used are CENTRAL and CLUSTER. To specify other operator classes, use the ALERT OPERATOR clause.

The NOTIFY IS ENABLED clause writes over any operator classes set by the RMU Set After_Journal Notify command.

The default is disabled.

This clause is available only on the OpenVMS platforms. ♦

**ALERT OPERATOR**

OpenVMS VAX OpenVMS Alpha

Specifies which operator will be notified of the occurrence of a catastrophic journaling event. You can specify the following operator classes:

| Operator Class | Meaning |
|---|---|
| ALL | The ALL operator class broadcasts a message to all terminals that are enabled as operators and that are attached to the system or cluster. These terminals must be turned on and have broadcast-message reception enabled. |
| NONE | The NONE operator class inhibits the display of messages to the entire system or cluster. |
| [NO] CENTRAL | The CENTRAL operator class broadcasts messages sent to the central system operator. The NO CENTRAL operator class inhibits the display of messages sent to the central system operator. |
| [NO] DISKS | The DISKS operator class broadcasts messages pertaining to mounting and dismounting disk volumes. The NO DISKS operator class inhibits the display of messages pertaining to mounting and dismounting disk volumes. |

| Operator Class | Meaning |
|---|---|
| [NO] CLUSTER | The CLUSTER operator class broadcasts messages from the connection manager pertaining to cluster state changes. The NO CLUSTER operator class inhibits the display of messages from the connection manager pertaining to cluster state changes. |
| [NO] SECURITY | The SECURITY operator class displays messages pertaining to security events. The NO SECURITY operator class inhibits the display of messages pertaining to security events. |
| [NO] OPER1 through [NO] OPER12 | The OPER1 through OPER12 operator classes display messages to operators identified as OPER1 through OPER12. The NO OPER1 through NO OPER12 operator classes inhibit messages from being sent to the specified operator. |

This clause is available only on the OpenVMS platforms. ♦

**OVERWRITE IS ENABLED**
**OVERWRITE IS DISABLED**

Specifies whether the overwrite option is enabled or disabled.

After-image journal files are used for database recovery in case of media failure and for transaction recovery as part of the fast commit feature. In some environments, only the fast commit feature is of interest and a small set of journal files can be used as a circular fast commit log with no backup of the contents required. The OVERWRITE option instructs Oracle Rdb to write over journal records that would normally be used for media recovery. The resulting set of journal files is unable to be used by the RMU Recover command for media recovery.

The OVERWRITE option is ignored when only one after-image journal (.aij) file exists. Adding subsequent journal files activates the OVERWRITE option.

The default is DISABLED.

**SHUTDOWN TIME IS n MINUTES**

Specifies the number of minutes the database system will wait after a catastrophic event before it shuts down the database. The shutdown time is the period, in minutes, between the point when the after-image journaling subsystem becomes unavailable and the point when the database is shut down. During the after-image journaling shutdown period, all database update activity is stalled.

If notification is enabled with the NOTIFY IS clause, operator messages will be broadcast to all enabled operator classes.

To recover from the after-image journaling shutdown state and to resume normal database operations, you must make an .aij file available for use. You can do this by backing up an existing modified journal file, or, if you have a journal file reservation available, by adding a new journal file to the after-image journaling subsystem. If you do not make a journal file available before the after-image journal shutdown time expires, the database will be shut down and all active database attachments will be terminated.

The after-image journaling shutdown period is only in effect when a fixed-size .aij file is used. When a single extensible .aij file is used, the default action is to shut down all database operations when the .aij file becomes unavailable.

The default is 60 minutes. The minimum value is 1 minute; the maximum value is 4320 minutes (3 days).

**alter-storage-area-params**
Parameters that change the characteristics of database storage area files. You can specify the same storage area parameters for either single-file or multifile databases, but the effect of the clauses in this part of an ALTER DATABASE statement differs.

- For single-file databases, the storage area parameters change the characteristics for the single storage area in the database.
- For multifile databases, the storage area parameters change the characteristics of the RDB$SYSTEM storage area.

  You can also change *some* of the characteristics of the RDB$SYSTEM storage area using the ALTER STORAGE AREA clause. However, you can only change the read-only and read/write parameters in this part of the ALTER DATABASE statement. See the ALTER STORAGE AREA clause later in this Arguments list for more information about the RDB$SYSTEM characteristics that you are allowed to alter.

The ALTER DATABASE statement does not let you change all storage area parameters you can specify in the CREATE DATABASE statement. You must use the EXPORT and IMPORT statements to change the following database root file parameters:

- INTERVAL
- PAGE FORMAT
- PAGE SIZE

- SNAPSHOT FILENAME
- THRESHOLDS

**ALLOCATION IS number-pages PAGES**
Specifies the number of database pages allocated to the storage area. The initial allocation never changes and is used for the hash algorithm. The new allocation becomes the current allocation. If you execute the RMU Dump /Header command, you see the initial and the current allocation.

SQL automatically extends the allocation to handle the storage requirements. Pages are allocated in groups of three (known as a *clump*). An ALLOCATION of 25 pages actually provides for 27 pages of data and subsequent expansion. The default is 400 pages.

The altered area is extended if the specified value exceeds the current area allocation. Otherwise the specified value is ignored.

**EXTENT ENABLED**
**EXTENT DISABLED**
Enables or disables extents. Extents are ENABLED by default and can be changed on line; however, the new extents are not immediately effective on all nodes of a cluster. On the node on which you have changed extents, the new storage area extents are immediately effective for all users. The new storage area extents become effective as the database is attached on each node of the cluster.

You can encounter performance problems when creating hashed indexes in storage areas with the mixed page format if the storage area was created specifying the wrong size for the area and if extents are enabled. By disabling extents, this problem can be diagnosed early and corrected to improve performance.

**EXTENT IS extent-pages PAGES**
**EXTENT IS (extension-options)**
Changes the number of pages of each storage area file extent. See the description under the SNAPSHOT EXTENT argument.

**MINIMUM OF min-pages PAGES**
Specifies the minimum number of pages of each extent. The default is 99 pages.

**MAXIMUM OF max-pages PAGES**
Specifies the maximum number of pages of each extent. The default is 9999 pages.

**PERCENT GROWTH IS growth**
Specifies the percent growth of each extent. The default is 20 percent growth.

**CACHE USING row-cache-name**
Assigns the named row cache area as the default for all storage areas in the database. All rows stored in this area, whether they consist of table data, segmented string data, or special rows such as index nodes, are cached.

The row cache area must exist before terminating the ALTER DATABASE statement.

Alter the database and storage area to assign a new row cache area to override the database default row cache area. Only one row cache area is allowed for each storage area.

If you do not specify the CACHE USING clause or the NO ROW CACHE clause, NO ROW CACHE is the default for the database.

**NO ROW CACHE**
Specifies that the database default is not to assign a row cache area to all storage areas in the database. You cannot specify the NO ROW CACHE clause if you specify the CACHE USING clause.

Alter the storage area and name a row cache area to override the database default. Only one row cache area is allowed for each storage area.

If you do not specify the CACHE USING clause or the NO ROW CACHE clause, NO ROW CACHE is the default for the database.

**LOCKING IS ROW LEVEL**
**LOCKING IS PAGE LEVEL**
Specifies if locking is at the page or row level. This clause provides an alternative to requesting locks on records. The default is ROW LEVEL, which is compatible with previous versions of Oracle Rdb.

When many records are accessed in the same area and on the same page, the LOCKING IS PAGE LEVEL clause reduces the number of lock operations perfomed to process a transaction; however, this is at the expense of reduced concurrency. Transactions that benefit most with page-level locking are of short duration and also access several database records on the same page.

Use the LOCKING IS ROW LEVEL clause if transactions are long in duration and lock many rows.

The LOCKING IS PAGE LEVEL clause causes fewer blocking asynchronous system traps and provides better response time and utilization of system resources. However, there is a higher contention for pages and increased potential for deadlocks.

Page-level locking is *never* applied to RDB$SYSTEM, either implicitly or explicitly, because the locking protocol can stall metadata users.

You cannot specify page-level locking on single-file databases.

**READ WRITE**
**READ ONLY**

The READ options of the alter-storage-area-params clause permit you to change existing storage area access as follows:

- Select the READ WRITE option to change any storage area to read/write access.
- Select the READ ONLY option to change any storage area to read-only access.

If you want to change the read-only and read/write parameters of the RDB$SYSTEM storage area, you must specify these parameters at this point of your ALTER DATABASE statement and not in the ALTER STORAGE AREA clause. For example:

```
SQL> -- You can change the RDB$SYSTEM storage area by altering
SQL> -- the database.
SQL> --
SQL> ALTER DATABASE FILENAME mf_personnel
cont> READ ONLY;
SQL> --
SQL> -- An error is returned if you try to change the RDB$SYSTEM storage
SQL> -- area to read-only using the ALTER STORAGE AREA clause.
SQL> --
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ALTER STORAGE AREA RDB$SYSTEM
cont> READ ONLY;
%RDB-E-BAD_DPB_CONTENT, invalid database parameters in the database
parameter block (DPB)
-RDMS-E-NOCHGRDBSYS, cannot change RDB$SYSTEM storage area explicitly
```

SQL provides support for read-only databases and databases with one or more read-only storage areas.

You can take advantage of read-only support if you have a stable body of data that is never (or rarely) updated. When the RDB$SYSTEM storage area is changed to read-only, lock conflicts occur less frequently, and the automatic updating of index and table cardinality is inhibited.

Read-only databases consist of:

- A read/write database root file
- One or more read-only storage areas and no read/write storage areas

Read-only databases can be published and distributed on CD–ROM.

Read-only storage areas:

- Have snapshot files but do not use them. (Data in a read-only storage area is not updated; specify a small number for the initial snapshot file size for a read-only storage area.)
- Eliminate page and record locking in the read-only storage areas.
- Are backed up by the RMU Backup command by default unless you explicitly state the Noread_Only qualifier, which excludes read-only areas without naming them.
- Are restored by the RMU Restore command if they were previously backed up.
- Are recovered by the RMU Recover command. However, unless the read-only attribute was modified, the read-only area does not change.
- Are not recovered by the RMU Recover command with the Area=* qualifier, in which you are not explicitly naming the areas needing recovery, unless they are inconsistent.

You use the READ ONLY option to change a storage area from read/write to read-only access. If you wanted to facilitate batch-update transactions to infrequently changed data, you would use the READ WRITE option to change a read-only storage area back to read/write.

If you change a read/write storage area to read-only, you cannot specify the EXTENT, SNAPSHOT ALLOCATION, and SNAPSHOT EXTENT clauses.

A database with both read/write and read-only storage areas can be fully recovered after a system failure *only* if after-image journaling is enabled on the database. If your database has both read/write and read-only storage areas but does not have after-image journaling enabled, perform full backup operations (including read-only areas) at all times. Doing full backup operations enables you to recover the entire database to its condition at the time of the previous backup operation.

For a complete description of read-only databases and read-only storage areas, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**SNAPSHOT ALLOCATION IS snp-pages PAGES**
Changes the number of pages allocated for the snapshot file. The default is 100 pages. If you have disabled the snapshot file, you can set the snapshot allocation to 0 pages.

**SNAPSHOT EXTENT IS extent-pages PAGES**
**SNAPSHOT EXTENT IS (extension-options)**
Changes the number of pages of each snapshot or storage area file extent. The default extent for storage area files is 100 pages.

Specify a number of pages for simple control over the file extent. For greater control, and particularly for multivolume databases, use the MINIMUM, MAXIMUM, and PERCENT GROWTH extension options instead.

If you use the MINIMUM, MAXIMUM, and PERCENT GROWTH parameters, you must enclose them in parentheses.

**CHECKSUM CALCULATION IS ENABLED**
**CHECKSUM CALCULATION IS DISABLED**
This option allows you to enable or disable calculations of page checksums when pages are read from or written to the storage area files.

The default is ENABLED.

**Note**

Oracle Rdb recommends that you leave checksum calculations enabled, which is the default.

With current technology, it is possible that errors may occur that the checksum calculation can detect but that may not be detected by either the hardware, firmware, or software. Unexpected application results and database corruption may occur if corrupt pages exist in memory or on disk but are not detected.

Oracle Rdb recommends performing checksum calculations, except in the following specific circumstances:

- Your application is stable and has run without errors on the current hardware and software configuration for an extended period of time.
- You have reached maximum CPU utilization in your current configuration. Actual CPU utilization by the checksum calculation depends primarily on the size of the database pages in your database. The larger the database page, the more noticeable the CPU usage by the checksum calculation may become.

**Note**

Oracle Rdb recommends that you carefully evaluate the trade-off between reducing CPU usage by the checksum calculation and the potential for loss of database integrity if checksum calculations are disabled.

Oracle Rdb allows you to disable and, subsequently, re-enable checksum calculation without error. However, once checksum calculations have been disabled, corrupt pages may not be detected even if checksum calculations are subsequently re-enabled.

**SNAPSHOT CHECKSUM CALCULATION IS ENABLED**
**SNAPSHOT CHECKSUM CALCULATION IS DISABLED**
Allows you to enable or disable calculations of page checksums when pages are read from or written to the snapshot files.

The default is ENABLED.

**Note**

Oracle Rdb recommends that you leave snapshot checksum calculations enabled, which is the default.

With current technology, it is possible that errors may occur that the snapshot checksum calculation can detect but that may not be detected by either the hardware, firmware, or software. Unexpected application results and database corruption may occur if corrupt pages exist in memory or on disk but are not detected.

Oracle Rdb recommends performing snapshot checksum calculations, except in the following specific circumstances:

- Your application is stable and has run without errors on the current hardware and software configuration for an extended period of time.
- You have reached maximum CPU utilization in your current configuration. Actual CPU utilization by the snapshot checksum calculation depends primarily on the size of the database pages in your database. The larger the database page, the more noticeable the CPU usage by the snapshot checksum calculation may become.

**Note**

Oracle Rdb recommends that you carefully evaluate the trade-off between reducing CPU usage by the snapshot checksum calculation and the potential for loss of database integrity if snapshot checksum calculations are disabled.

---

Oracle Rdb allows you to disable and, subsequently, re-enable snapshot checksum calculation without error. However, once snapshot checksum calculations have been disabled, corrupt pages may not be detected even if snapshot checksum calculations are subsequently re-enabled.

**ADD CACHE row-cache-name**
Creates a row cache area.

**ALLOCATION IS n BLOCK**
**ALLOCATION IS n BLOCKS**
Specifies the initial allocation of the row cache file (.rdc) to which cached rows are written.

If the ALLOCATION clause is not specified, the default allocation in blocks is approximately 40 percent of the CACHE SIZE for this cache.

**EXTENT IS n BLOCK**
**EXTENT IS n BLOCKS**
Specifies the file extent size for the row cache file (.rdc).

If the EXTENT clause is not specified, the default number of blocks is CACHE SIZE * 127 for this cache.

**CACHE SIZE IS n ROW**
**CACHE SIZE IS n ROWS**
Specifies the number of rows allocated to the row cache area. As the row cache area fills, rows more recently referenced are retained in the row cache area while those not referenced recently are discarded. Adjusting the allocation of the row cache area helps to retain important rows in memory. If not specified, the default is 1000 rows.

The product of the CACHE SIZE and the ROW LENGTH settings determines the amount of memory required for the row cache area. (Some additional overhead and rounding up to page boundaries is performed by the database system.) The row cache area is shared by all processes attached to the database.

**LARGE MEMORY IS ENABLED**
**LARGE MEMORY IS DISABLED**

OpenVMS Alpha

Specifies whether or not large memory is used to manage the row cache. Very large memory (VLM) allows Oracle Rdb to use as much physical memory as is available. It provides access to a large amount of physical memory through small virtual address windows.

Use LARGE MEMORY IS ENABLED only when both of the following are true:

- You have enabled row caching.
- You want to cache large amounts of data, but the cache does not fit in the virtual address space.

The default is DISABLED. See the Usage Notes for restrictions pertaining to the very large memory (VLM) feature. ♦

**ROW REPLACEMENT IS ENABLED**
**ROW REPLACEMENT IS DISABLED**

Specifies whether or not Oracle Rdb replaces rows in the cache. When the ROW REPLACEMENT IS ENABLED clause is used, rows are replaced when the row cache area becomes full. When the ROW REPLACEMENT IS DISABLED clause is used, rows are not replaced when the cache is full. The type of row replacement policy depends upon the application requirements for each cache.

The default is ENABLED.

**LOCATION IS directory-spec**

Specifies the name of the directory to which row cache information is written. The database system generates a file name (row-cache-name.rdc) automatically for each row cache area at checkpoint time. Specify a device name and directory name only enclosed within by single quotation marks. The file name is the row-cache-name specified when creating the row cache area. By default, the location is the directory of the database root file. These .rdc files are permanent database files.

This LOCATION clause overrides a previously specified location at the database level.

**NO LOCATION**

Removes the location previously specified in a LOCATION IS clause for the row cache area. If you specify NO LOCATION, the row cache location becomes the directory of the database root file.

**NUMBER OF RESERVED ROWS IS n**
Specifies the maximum number of cache rows that each user can reserve. The default is 20 rows.

**ROW LENGTH IS n BYTE**
**ROW LENGTH IS n BYTES**
Specifies the size of each row allocated to the row cache area. Rows are not cached if they are too large for the row cache area. The ROW LENGTH is an aligned longword rounded up to the next multiple of 4 bytes.

The maximum row length in a row cache area is 65535 bytes.

If the ROW LENGTH clause is not specified, the default row length is 256 bytes.

**SHARED MEMORY IS SYSTEM**
**SHARED MEMORY IS PROCESS**
OpenVMS Alpha ☰ Determines whether cache global sections are created in system space or process space. The default is SHARED MEMORY IS PROCESS.

When you use cache global sections created in the process space, you and other users share physical memory and the OpenVMS Alpha operating system maps a row cache area to a private address space for each user. As a result, all users are limited by the free virtual address range and each use a percentage of memory in overhead. If many users are accessing the database, the overhead can be high.

When many users are accessing the database, consider using SHARED MEMORY IS SYSTEM. This gives users more physical memory because they share the system space of memory and there is none of the overhead associated with the process space of memory. ♦

**WINDOW COUNT IS n**
OpenVMS Alpha ☰ Specifies the number of virtual address windows used by the LARGE MEMORY clause.

The window is a view into the physical memory used to create the very large memory (VLM) information. Because the VLM size may be larger than that which can be addressed by a 32-bit pointer, you need to view the VLM information through small virtual address windows.

You can specify a positive integer in the range from 10 through 65535. The default is 100 windows. ♦

**ADD JOURNAL journal-name**
Creates a new journal file.

**FILENAME journal-file-spec**
Specifies the journal file specification with the default file extension .aij.

**ALLOCATION IS n BLOCKS**
Specifies the number of blocks allocated for the .aij file. The default and minimum is 512 blocks.

**EXTENT IS n BLOCKS**
Specifies the number of blocks of each .aij file extent. The default and minimum extent for .aij files is 512 blocks.

**BACKUP FILENAME backup-file-spec**
Specifies the file specification to be used by the backup server.

During execution, the backup server and the RMU Backup After_Journal command use this file specification as the name of the backup file. You can override this value by specifying a file name for the journal file using the RMU Backup After_Journal command.

**backup-filename-options**
See the backup-filename-options earlier in this Arguments list for details.

**SAME BACKUP FILENAME AS JOURNAL**
See the argument described earlier in this section for information about this clause.

**NO BACKUP FILENAME**
Removes a previously established backup file name specification.

**ADD STORAGE AREA area-name FILENAME file-spec**
Specifies the name and file specification for a storage area you want to add to the database. You can use the ADD STORAGE AREA clause only on multifile databases. The storage area name cannot be the same as any other storage area name in the database.

The ADD STORAGE AREA clause creates two files: a storage area file with an .rda file extension and a snapshot file with an .snp file extension. If you omit the FILENAME argument, the file specification uses the following defaults:

- Device—the current device for the process (on OpenVMS only)
- Directory—the current directory for the process
- File name—the name specified for the storage area

The file specification is used for the storage area and snapshot files that comprise the storage area (unless you use the SNAPSHOT FILENAME argument to specify a different file for the snapshot file, which you can only specify with a multifile database). Because the ADD STORAGE AREA clause may create two files with different file extensions, do not specify a file extension with the file specification.

If you use the ALTER DATABASE statement to add a storage area, the change is journaled, however, you should back up your database before making such a change.

For important information about changes that are not journaled, see the Usage Notes.

**storage-area-params-1**
**storage-area-params-2**
Parameters that control the characteristics of the storage area. For more information on the parameters, see the CREATE STORAGE AREA Clause.

**ALTER CACHE row-cache-name**
Alters existing row cache areas.

**row-cache-params**
For information regarding the row-cache-params, see the descriptions under the ADD CACHE argument described earlier in this arguments list.

**ALTER JOURNAL journal-name**
Alters existing journal files. RDB$JOURNAL is the default journal name if no name is specified.

**JOURNAL IS UNSUPPRESSED**
If a journal file becomes inaccessible, it is disabled by the journaling system. It remains in that state until you correct the problem and manually unsuppress that journal file.

**BACKUP FILENAME backup-file-spec**
Alters the name of the backup file used by the backup server.

**backup-filename-options**
See the backup-filename-options earlier in this Arguments list for details.

**SAME BACKUP FILENAME AS JOURNAL**
Allows you to alter the name assigned to the backup file when you alter the name of the journal file.

**NO BACKUP FILENAME**
Removed a previously established backup file name specification.

**ALTER STORAGE AREA area-name**
Specifies the name of an existing storage area in the database that you want to alter. You can use the ALTER STORAGE AREA clause only on multifile databases.

You can specify RDB$SYSTEM for the area-name if you are altering the following clauses:

- ALLOCATION IS number-pages PAGES
- extent-params
- CACHE USING row-cache-name
- NO ROW CACHE
- SNAPSHOT ALLOCATION IS snp-pages PAGES
- SHAPSHOT EXTENT
- CHECKSUM CALCUALTION
- SNAPSHOT CHECKSUM CALCULATION

Oracle Rdb generates an error if you specify RDB$SYSTEM as the area-name when altering the following clauses:

- LOCKING IS PAGE LEVEL
- READ WRITE
- READ ONLY
- WRITE ONCE

If you want to change the read-only and read/write parameters of the RDB$SYSTEM storage area using the ALTER DATABASE statement, you must specify these parameters outside of the ALTER STORAGE AREA clause. For example:

```
SQL> -- You can change the RDB$SYSTEM storage area by altering the
SQL> -- database.
SQL> --
SQL> ALTER DATABASE FILENAME mf_personnel
cont> READ ONLY;
SQL> --
SQL> -- An error is returned is you try to change the RDB$SYSTEM storage
SQL> -- area to read-only using the ALTER STORAGE AREA clause.
SQL> --
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ALTER STORAGE AREA RDB$SYSTEM
cont> READ ONLY;
%RDB-E-BAD_DPB_CONTENT, invalid database parameters in the database
parameter block (DPB)
-RDMS-E-NOCHGRDBSYS, cannot change RDB$SYSTEM storage area explicitly
```

You cannot alter the RDB$SYSTEM storage area to write once.

**alter-storage-area-params**
Parameters that the ALTER STORAGE AREA clause changes. See alter-storage-area-params earlier in this section (after the SHUTDOWN TIME IS n MINUTES argument) for details about the parameters.

**WRITE ONCE**
The WRITE ONCE option of the ALTER STORAGE AREA clause permits you to create a storage area that contains only a segmented string in a format that can be stored on a write-once, read-many (WORM) optical device. This option can only be used with the ADD STORAGE AREA and ALTER STORAGE AREA clauses.

You can use the WRITE ONCE option to change a storage area containing stable database list (segmented string) data to a format that can be stored on a write-once, read-many (WORM) optical device. A WORM optical device offers a relatively inexpensive way of storing large amounts of data for read-only access compared to other storage media. Oracle Rdb supports the Perceptics WORM optical disk drive and jukebox as a storage media for storing lists or segmented string data. Example 2 at the end of this section shows how to change a read/write storage area to a write-once storage area.

Oracle Rdb permits the storing of many write-once list segments on one write-once page, resulting in better write-once space usage. This improves storage performance because the storage algorithm reduces I/O due to more compact storage.

The following restrictions apply to the WRITE ONCE option:

- You cannot write data other than segmented strings to a write-once storage area. SQL issues an error message if you try to create a storage map that stores data other than segmented strings in a write-once storage area. Storage maps for nonsegmented string data must be removed before you can alter a storage area to WRITE ONCE.
- When you create a storage area on WORM media, you must specify that the snapshot area remains on a read/write device; do not give a snapshot file the WRITE ONCE attribute.
- If you specify the WRITE ONCE option when storing a segmented string, database keys are not compressed. For more information on database key compression, see the *Oracle Rdb7 Guide to Database Maintenance*.
- WORM areas do not use SPAM pages. However, to assist moving data back to non-WORM devices on which SPAM pages must be built again, space is still allocated for them. Because SPAM pages are essential in uniform areas, write-once storage areas cannot be of uniform format and, therefore, are required to be of mixed format.
- You can use the PAGE SIZE IS clause of the CREATE DATABASE statement to set the default page size for a storage area. To optimize storage, always specify an even number of blocks per page for a write-once storage area.
- Oracle Rdb does not support magnetic media for storing write-once storage areas.
- After you move a storage area to or from WORM media, back up your database completely and start a new after-image journal file. For more information on backup and recovery procedures with write-once storage areas, see the *Oracle Rdb7 Guide to Database Maintenance*.
- The storage algorithm does not attempt to compute the best fit for write-once list segments.
- The storage algorithm does not allow write-once storage by different users to be on the same write-once page.
- If the number of buffers is small, a write-once page that is only partially full may be transferred out of the buffer pool (and hence written to disk) as part of the usual buffer replacement policy.
- You can only specify the WRITE ONCE option with the ADD STORAGE AREA and ALTER STORAGE AREA clauses.

**JOURNAL IS ENABLED**
**JOURNAL IS DISABLED**
Specifies whether or not WRITE ONCE areas are written to the .aij file.

Disabling the journaling attribute on WRITE ONCE areas is beneficial because after-image journaling on storage media can slow the loading of large images or exceed storage area availability.

However, if there is a failure of the storage media, there may be loss of space or, more important, loss of information. In the case of a magnetic disk failure, the database is restored from an earlier backup and the AIJ records are applied to the restored database. There is no loss of information in this case, but there could be loss of space because list of byte varying data written before the failure is not referenced by the existing data rows, and these list column values take up space on the write-once media that cannot be reused.

In the case of a WORM device failure, there can be loss of information because the existing data rows reference list column data that is no longer available. For example, if 120 pages were allocated in the WRITE ONCE area and 100 pages had data written to the them, and the last backup was done when the area had 50 pages of information, any data on pages 51 to 120 is lost if there is a failure of the WORM device. Pages 51 to 120 are inaccessible. The RMU Repair command can be used to repair rows that reference missing list column data. For more information, see the *Oracle Rdb7 Guide to Database Maintenance* and the *Oracle RMU Reference Manual*.

Remember, the write-once storage area must be of mixed format.

The default is JOURNAL IS ENABLED.

**DROP CACHE row-cache-name CASCADE**
**DROP CACHE row-cache-name RESTRICT**
Deletes the specified row cache area from the database.

If the mode is RESTRICT, an exception is raised if the row cache area is assigned to a storage area.

If the mode is CASCADE, the row cache area is removed from all referencing storage areas.

The default is RESTRICT if no mode is specified.

**DROP STORAGE AREA area-name CASCADE**
**DROP STORAGE AREA area-name RESTRICT**
Deletes the specified storage area definition and the associated storage area and snapshot files. You can use the DROP STORAGE AREA clause only on multifile databases.

If you use the RESTRICT keyword, you cannot delete a storage area if any database object, such as a storage map, refers to the area or if there is data in the storage area.

If you use the CASCADE keyword, Oracle Rdb modifies all objects that refer to the storage area so that they no longer refer to it. However, Oracle Rdb does not delete objects if doing so makes the database inconsistent.

If you use the ALTER DATABASE statement to delete a storage area, the change is journaled, however, you should back up your database before making such a change.

See the Usage Notes for additional information on deleting storage areas or for important information about changes that are not journaled.

**DROP JOURNAL journal-name**
Deletes the specified journal file from the database.

You can only delete an .aij file that is not current and that has been backed up.

## Usage Notes

- Some database or storage area characteristics can be changed while users, including yourself, are attached to the database. See Table 6–2 for more information regarding the database-wide parameters you can modify while other users are attached to the database. If the characteristic you want to change cannot be changed while the database is being accessed, you will get the following error message:

  ```
  SQL> ATTACH 'FILENAME personnel';
  SQL> ALTER DATABASE FILENAME personnel MULTISCHEMA IS ON;
  %RDB-E-LOCK_CONFLICT, request failed due to locked resource
  -RDMS-F-LCKCNFLCT, lock conflict on client
  SQL> DISCONNECT DEFAULT;
  SQL> ALTER DATABASE FILENAME personnel MULTISCHEMA IS ON;
  ```

  If users are attached to the database when you change a characteristic, some changes are not visible to those users until they detach and reattach to the database.

  For more information regarding database characteristics that can and cannot be changed on line, see the *Oracle Rdb7 Guide to Database Design and Definition*.

- The ALTER DATABASE statement is not executed in a transaction context and, therefore, its effects are immediate and cannot be rolled back or committed.

- You cannot delete a storage area if it is the DEFAULT STORAGE AREA, the DEFAULT LIST STORAGE AREA, or the RDB$SYSTEM storage area.
- If you or another user are attached to the database when you add or delete a storage area, you get a file access conflict error message.

  For more information on restrictions to adding and deleting storage areas, see the *Oracle Rdb7 Guide to Database Design and Definition*.
- Keep the following in mind when deleting a storage area using CASCADE:
  - If the storage area is the only area in the storage map, Oracle Rdb deletes the storage area and all referencing objects.
  - If the storage map that refers to the area is strictly partitioned, Oracle Rdb deletes the storage area and all referencing objects, even if the storage map refers to more than one area.
  - If the storage area contains only an index, Oracle Rdb does not delete the area because doing so makes the database inconsistent.
  - If a hashed index and a table are in the same storage area and the mapping for the hashed index is not the same as the mapping for the table, Oracle Rdb does not delete the storage area.
  - If a storage area contains a table that contains constraints, Oracle Rdb only deletes the area if after doing so, the database remains consistent.
- When the LOCKING IS PAGE LEVEL or LOCKING IS ROW LEVEL clause is specified at the database level (using the ALTER DATABASE or CREATE DATABASE statements), all storage areas are affected (with the exception of RDB$SYSTEM, which is always set to row-level locking).
- You cannot disable journaling for read/write storage areas.
- SQL does not journal metadata updates for the following changes to the database parameters:
  - Changing the number of users
  - Changing the number of nodes
  - Reserving slots for journal files
  - Reserving slots for storage areas

  Unlike most metadata updates, database and storage area updates complete with an implicit commit operation. This means that you will not be able to issue a ROLLBACK statement if you make an error in your ALTER DATABASE statement.

**Note**

If you plan to change any of the database parameters that are not journaled, Oracle Rdb recommends that you back up your database before attempting these changes. If a change that is not journaled fails for some reason, the database becomes corrupt. If you backed up your database, you can restore your database from the backup copy.

- See the *Oracle Rdb7 Guide to Database Design and Definition* for a complete discussion of when to use the IMPORT, EXPORT, and ALTER DATABASE statements.
- Table 6–1 show which data definitions can be updated while users are attached to the database. For more information and restrictions not included in the Comments column of this table, see the *Oracle Rdb7 Guide to Database Design and Definition* and the *Oracle RMU Reference Manual*.

**Table 6–1 Updating Data Definitions While Users Are Attached to the Database**

| Metadata Update | Concurrency Allowed[1] | Comments |
|---|---|---|
| Catalogs<br>CREATE<br>DROP | Yes | You cannot delete a catalog when there are active transactions that access the catalog. |
| Collating sequences<br>CREATE<br>ALTER<br>DROP | Yes | You cannot delete a collating sequence if the database or domain in the database uses that collating sequence. |
| Constraints<br>CREATE<br>DROP | Yes | You cannot delete a constraint when there are active transactions that access the tables involved. |
| Domains<br>CREATE<br>ALTER<br>DROP | Yes | You cannot alter a domain if stored routines use the domain. |

[1]*Concurrency Allowed* means other users can attach to the database while the metadata update is being performed. Note that other restrictions, as described in the Comments column of this table, may apply.

(continued on next page)

**Table 6–1 (Cont.) Updating Data Definitions While Users Are Attached to the Database**

| Metadata Update | Concurrency Allowed[1] | Comments |
|---|---|---|
| External routines<br>CREATE<br>DROP | Yes | Refers to external procedures and functions. |
| Indexes<br>CREATE<br>ALTER<br>DROP | Yes | You cannot disable an index or delete an index definition when there are active transactions that access the tables involved. |
| Modules<br>CREATE<br>DROP | Yes | Modules contain stored procedures and functions. |
| Outlines<br>CREATE<br>DROP | Yes | |
| Protection<br>GRANT<br>REVOKE | Yes | Granting or revoking a privilege takes effect after the user detaches and attaches to the database again. |
| Schemas<br>CREATE<br>DROP | Yes | You cannot delete a schema when there are active transactions that access the schema. |
| Storage areas<br>RESERVE | No | This change is not journaled. |
| CREATE<br>ADD<br>DROP | Yes | Concurrency is allowed if the database root file contains available slots; that is, slots that have been reserved for storage areas but not used. Updates are not seen by users currently attached to the database. New areas are seen when new users attach to the database after the change is committed. These metadata operations complete with an implicit commit operation. |
| ALTER | See comments | You can modify many of the storage area parameters. See Table 6–2 for specific information. |
| Storage maps<br>CREATE<br>ALTER<br>DROP | Yes | |

[1] *Concurrency Allowed* means other users can attach to the database while the metadata update is being performed. Note that other restrictions, as described in the Comments column of this table, may apply.

(continued on next page)

**Table 6–1 (Cont.) Updating Data Definitions While Users Are Attached to the Database**

| Metadata Update | Concurrency Allowed[1] | Comments |
|---|---|---|
| Tables<br>CREATE<br>ALTER<br>DROP<br>TRUNCATE | Yes | You cannot delete a table definition when there are active transactions that use the table. |
| Triggers<br>CREATE<br>DROP | Yes | You cannot delete a trigger definition when there are active transactions that use the trigger or that refer to the tables involved. |
| Views<br>CREATE<br>DROP | Yes | Deleting a view does not affect active users until you commit your transaction, users detach from the database, and then attach to the database again. |
| Databases<br>CREATE<br>DROP | No | These metadata updates complete with an implicit commit operation. If a user is attached to the database when you attempt to delete a database, you receive the -SYSTEM-W-ACCONFLICT, file access conflict error message. |
| ALTER | See comments | You can modify many of the database parameters, including storage area parameters. See Table 6–2 for specific information. |

[1]*Concurrency Allowed* means other users can attach to the database while the metadata update is being performed. Note that other restrictions, as described in the Comments column of this table, may apply.

- Table 6–2 shows which database-wide parameters you can modify while other users are attached to the database. Remember that you cannot create or delete a database while any users are attached to it, including yourself. See the *Oracle Rdb7 Guide to Database Design and Definition* and the *Oracle RMU Reference Manual* for additional information and restrictions not included in the Comments column of this table.

**Table 6–2 Updating to Database-Wide Parameters While Users Are Attached to the Database**

| Metadata Update | On Line [1] | Comments |
|---|---|---|
| **Root File Parameters** | | |
| Open mode | Yes | Updates are not seen until a database open operation is required. |
| Wait interval for close | Yes | Updates do not take effect until the database is opened again after the change is completed. However, updates are not seen by users who attached to the database before the update. |
| Number of users | No | This change is not journaled. |
| Number of nodes | No | This change is not journaled. |
| Buffer size | No | |
| Number of buffers | Yes | Updates are not seen by users currently attached to the database. Updates are seen when new users attach to the database after the change is completed. |
| Number of recovery buffers | Yes | Updates take effect when a new database recovery process begins. |
| Recovery-unit journal location | Yes | |
| Global buffers enabled or disabled | No | |
| Number of global buffers | Yes | Updates do not take effect until the database is opened again after the change is completed. However, updates are not seen by users who attached to the database before the update. |
| Maximum number of global buffers per user | Yes | Updates do not take effect until the database is opened again after the change is completed. However, updates are not seen by users who attached to the database before the update. |
| Page transfer | Yes | Updates are not seen by users currently attached to the database. Updates are seen when new users attach to the database after the change is completed. |
| Adjustable lock granularity | No | |
| Carry-over locks enabled or disabled | No | |

[1]*On Line* means other users can attach to the database while the metadata update is being performed. Other restrictions, as described in the Comments column of this table, may apply.

(continued on next page)

**Table 6–2 (Cont.) Updating to Database-Wide Parameters While Users Are Attached to the Database**

| Metadata Update | On Line [1] | Comments |
|---|---|---|
| **Root File Parameters** | | |
| Lock timeout interval | Yes | Updates are not seen by users currently attached to the database. Updates are seen when new users attach to the database after the change is completed. |
| Statistics enabled or disabled | No | |
| Cardinality collection enabled or disabled | Yes | |
| Workload collection enabled or disabled | Yes | |
| Asynchronous batch-writes | Yes | Updates are not seen by users currently attached to the database. Updates are seen when new users attach to the database after the change is completed. |
| Asynchronous prefetch | Yes | Updates are not seen by users currently attached to the database. Updates are seen when new users attach to the database after the change is completed. |
| Detected asynchronous prefetch | Yes | Updates are not seen by users currently attached to the database. Updates are seen when new users attach to the database after the change is completed. |
| Incremental backup | Yes | |
| Lock partitioning | No | |
| Metadata changes enabled or disabled | Yes | Updates are not seen by users currently attached to the database. Updates are seen when new users attach to the database after the change is completed. |
| Checksum calculation | No | |
| Reserve row cache slots | No | This change is not journaled. |
| Row cache enabled or disabled | No | This change is not journaled. |
| Create or add row cache | Yes | |
| Alter row cache | No | |
| Delete row cache | No | |

[1] *On Line* means other users can attach to the database while the metadata update is being performed. Other restrictions, as described in the Comments column of this table, may apply.

(continued on next page)

**Table 6–2 (Cont.) Updating to Database-Wide Parameters While Users Are Attached to the Database**

| Metadata Update | On Line [1] | Comments |
|---|---|---|
| **Root File Parameters** | | |
| Row cache attributes | No | |
| Snapshot files enabled or disabled | No | |
| Snapshot files immediate or deferred | No | |
| Snapshot checksum calculation | No | |
| Reserve journal | No | This change is not journaled. |
| Journaling enabled or disabled | No | |
| Add journal | Yes | Online changes are allowed if the database root file contains available slots; that is, slots that have been reserved for journal files but not used. |
| Alter journal | Yes | |
| Delete journal | Yes | You cannot delete a journal file while it is in use. |
| Journal name or file name | No | |
| Journal allocation | Yes | |
| Journal backup server | Yes | |
| Journal backup file name | Yes | |
| Journal backup file name edit string | Yes | |
| Journal cache file name | Yes | |
| Journal extent | Yes | |
| Journal fast commit | No | |
| Journal checkpoint interval | No | |
| Journal checkpoint time | No | |

[1]*On Line* means other users can attach to the database while the metadata update is being performed. Other restrictions, as described in the Comments column of this table, may apply.

(continued on next page)

**Table 6–2 (Cont.) Updating to Database-Wide Parameters While Users Are Attached to the Database**

| Metadata Update | On Line [1] | Comments |
|---|---|---|
| **Root File Parameters** | | |
| Journal commit to journal optimization | No | |
| Journal transaction interval | No | |
| Journal log server | Yes | |
| Journal notify | Yes | |
| Journal overwrite | Yes | |
| Journal shutdown time | Yes | |
| **Storage Area Parameters** | | |
| Reserve storage area | No | This change is not journaled. |
| Specify default storage area | Yes | |
| Read or write attribute | Yes | Requires exclusive access to the area. |
| Journaling enabled or disabled for write-once areas | No | |
| Allocation | Yes | |
| Extension enabled or disabled | Yes | Updates are not seen by users currently attached to the database. Updates are seen when new users attach to the database after the change is completed. |
| Extension options | Yes | |
| Lock-level options | No | |
| Thresholds | Yes | Requires exclusive access to the area. |
| Snapshot file allocation | Yes | Truncating snapshot file blocks read-only transactions. |
| Snapshot checksum allocation | No | |

[1]*On Line* means other users can attach to the database while the metadata update is being performed. Other restrictions, as described in the Comments column of this table, may apply.

(continued on next page)

**Table 6–2 (Cont.) Updating to Database-Wide Parameters While Users Are Attached to the Database**

| Metadata Update | On Line [1] | Comments |
|---|---|---|
| **Storage Area Parameters** | | |
| Snapshot file extension options | Yes | |
| SPAMs enabled or disabled | Yes | Requires exclusive access to the area. Use the RMU qualifiers Spams or Nospams. |
| Checksum calculation | No | |
| **Security Parameters** | | |
| Audit file name | Yes | Use the RMU Set Audit command. |
| Alarm name | Yes | Use the RMU Set Audit command. |
| Audit enabled or disabled | Yes | Use the RMU Set Audit command. |
| Alarm enabled or disabled | Yes | Use the RMU Set Audit command. |
| Audit FIRST flag | Yes | Use the RMU Set Audit command. |
| Audit FLUSH flag | Yes | Use the RMU Set Audit command. |
| Audit event class flags | Yes | Use the RMU Set Audit command. |

[1]*On Line* means other users can attach to the database while the metadata update is being performed. Other restrictions, as described in the Comments column of this table, may apply.

- You cannot specify a snapshot file name for a single-file database.

  The SNAPSHOT FILENAME clause specified outside the CREATE STORAGE AREA clause is used to provide a default for subsequent CREATE STORAGE AREA statements. Therefore, this clause does not allow you to create a separate snapshot file for a single-file database (a database without separate storage areas).

  When you create a single-file database, Oracle Rdb does not store the file specification of the snapshot file. Instead, it uses the file specification of the root file (.rdb) to determine the file specification of the snapshot file.

  If you want to place the snapshot file on a different device or directory, Oracle Rdb recommends that you create a multifile database.

OpenVMS VAX OpenVMS Alpha However, you can work around the restriction on OpenVMS platforms by defining a search list, for a concealed logical name. (However, do not use a nonconcealed rooted logical. Database files defined with a nonconcealed rooted logical can be backed up, but do not restore as expected.)

To create a database with a snapshot file on a different device or directory:

1. Define a search list using a concealed logical name. Specify the location of the root file as the first item in the search list and the location of the snapshot file as the second item.
2. Create the database using the logical name for the directory specification.
3. Copy the snapshot file to the second device or directory.
4. Delete the snapshot file from the original location.

If you are doing this with an existing database, close the database using the RMU Close command before defining the search list, and open the database using the RMU Open command after deleting the original snapshot file. Otherwise, follow the preceding steps.

An important consideration when placing snapshot and database files on different devices is the process of backing up and restoring the database. Use the RMU Backup command to back up the database. You can then restore the files by executing the RMU Restore command. Copy the snapshot file to the device or directory where you want it to reside, and delete the snapshot file from the location to which it was restored. For more information, see the *Oracle RMU Reference Manual*. ♦

- To move the database root file, storage areas, and snapshot files to different disks, use the RMU Move_Area command. To move database files to another system, use the RMU Backup and RMU Restore commands. For more information about Oracle RMU commands, see the *Oracle RMU Reference Manual*.
- An exception message is returned if the RDB$SYSTEM storage area is read-only and you try to ready a table in exclusive or batch-update mode.

  Exclusive access to a table or index must always write to the RDB$SYSTEM storage area because this type of access does not write the “before” images of the modified data into the snapshot file. Consequently, a read-only access to the same table or index must have a way of knowing whether or not the snapshot file can produce the data it requires.

Each exclusive access must record that it is not maintaining snapshots on a per index or per table basis, as this is the unit of data for which Oracle Rdb permits the setting of the access mode. The natural location to store the fact that snapshots are not being maintained is with the table or index definition because the definition must be accessed when the table or index is reserved. Storing it elsewhere incurs additional overhead.

The table and index definitions are stored in the RDB$SYSTEM area. Consequently, if the RDB$SYSTEM area is set to read-only, you are not permitted to access any table or index in the exclusive mode. This condition affects all database access.

- If your database has snapshots set to ENABLED DEFERRED, users may not be able to attach to the database once you issued one of the following statements:
  - CREATE, ALTER, or DROP TABLE
  - CREATE or DROP INDEX

  During a database attach, Oracle Rdb locks certain key metadata tables and reads them to construct the metadata information cache used to process requests against the database. When one of the previously listed statements executes a read/write transaction that updates these metadata tables, any subsequent database attach (equivalent to a read-only transaction) stalls until the read/write transaction is completed. Users that were attached to the database before the statement was issued can continue accessing the database.

  Use of deferred snapshots cause conflict when using data definition language (DDL) statements in a production environment because snapshot copies of the system metadata cannot be written to the snapshot file.

  To avoid this problem, modify the database so that snapshots are set to ENABLED IMMEDIATE. You can use any of the following statements to set snapshots to ENABLED IMMEDIATE:
  - CREATE DATABASE
  - ALTER DATABASE
  - IMPORT
- Oracle Rdb uses the extensible after-image journaling feature as the default until you specifically add another journal file.
- Adding one journal file to an existing extensible journal file automatically converts it to a fixed-size journal file. See the *Oracle Rdb7 Guide to Database Design and Definition* for additional information.

- Because the creation of a journal file does not cause an immediate switch of journal files, Oracle Rdb recommends that you do not delete journal files.
- Oracle Rdb recommends that each .aij file be located on devices separate from each other and from other database files so that you can recover from a hardware or software failure.
- Exclusive database access is required for the following operations:
  - Reserving after-image journal files
  - Enabling after-image journal files
  - Disabling after-image journal files
  - Reserving storage areas
- You do not need exclusive database access to add, delete, or alter .aij files or storage areas.

  However, when you add a storage area with a page size that is smaller than the smallest storage area page size, you must have exclusive access to the database.
- The system allows you to disable journaling, reserve additional slots, and then continue processing without re-enabling the journaling feature. If you do this, the system tells you that your database is not recoverable. Be sure to enable journaling before any further processing.
- Use the SHOW statement or the RMU Dump command with the Header qualifier to review your current journaling and storage area status.
- Use the RMU Backup command to back up the database.
- There is no tape support for the AIJ backup server (ABS).
- Adding and deleting storage areas are online operations (not requiring exclusive database access). Reserving storage area slots is an offline operation (requiring exclusive database access). Therefore, you cannot specify an ADD or DROP STORAGE AREA clause and a RESERVE STORAGE AREA clause in the same ALTER DATABASE statement. For example:

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> RESERVE 2 STORAGE AREAS
cont> ADD STORAGE AREA TEST_ONE
cont>    FILENAME mf_pers_test;
%RDB-E-BAD_DPB_CONTENT, invalid database parameters in the database
parameter block (DPB)
-RDMS-E-CONFRESERVE, RESERVE cannot be used with ADD/DROP in the same
ALTER DATABASE command
```

- If you specify the WRITE ONCE (JOURNAL IS DISABLED) clause, a database that is recovered to a time prior to all transactions being committed causes old list of byte varying data to be visible again. If the database is recovered using a backup copy, access to some list of byte varying columns returns an exception to indicate that old data is present on the write-once media.
- Use one of the following Oracle RMU commands to change some of the root characteristics of a single-file database that can be directly altered for a multifile database:
  - Restore
  - Copy
  - Move_Area
- ADD CACHE and ALTER CACHE clauses do not assign the row cache area to a storage area. You must use the CACHE USING clause with the CREATE STORAGE AREA clause of the CREATE DATABASE statement or the CACHE USING clause with the ADD STORAGE AREA or ALTER STORAGE AREA clauses of the ALTER DATABASE statement.
- The product of the CACHE SIZE and the ROW LENGTH settings determines the amount of memory required for the row cache area (some additional overhead and rounding up to page boundaries is performed by the database system). The row cache area is shared by all processes attached to the database from any node.
- The row cache area is shared by all processes attached to the database on any node.
- The following are requirements when using the row caching feature:
  - Fast commit must be enabled
  - Number of cluster notes must equal 1
- When you alter the row length of a row cache area, Oracle Rdb rounds the specified value up to the next value divisible by four. For example, if you alter the row length to 30, Oracle Rdb assigns 32. This is done because longword aligned structures are more optimal for memory access.
- The DICTIONARY IS REQUIRED flag is cleared if you specify the DICTIONARY IS NOT USED clause.
- You must use the FILENAME clause, and not the PATHNAME clause, when removing the link between the repository and the database with the DICTIONARY IS NOT USED clause.

- The EDIT STRING options to the BACKUP FILENAME clause are appended to the backup file name in the order in which you specify them. For example, the following portion of syntax creates an OpenVMS file with the name BACKUP160504233.AIJ when journal 3 is backed up at 4:05 in the afternoon on April 23.

  ```
     .
     .
     .
  cont> BACKUP FILENAME 'DISK2:[DIRECTORY2]BACKUP'
  cont>    (EDIT STRING IS HOUR + MINUTE + MONTH + DAY + SEQUENCE)
     .
     .
     .
  ```

  You can make the file name (BACKUP$1605_0423_3.AIJ) more readable by inserting string literals between each edit string option as shown in the following example:

  ```
     .
     .
     .
  cont> BACKUP FILENAME 'DISK2:[DIRECTORY2]BACKUP'
  cont>    (EDIT STRING IS '$' + HOUR + MINUTE + '_' +
  cont>                     MONTH + DAY + '_' + SEQUENCE)
     .
     .
     .
  SQL> SHOW JOURNAL BACKUP;
       BACKUP
          Journal File:    DISK1:[DIRECTORY1]BACKUP.AIJ;1
          Backup File:     DISK2:[DIRECTORY2]BACKUP.AIJ;
          Edit String:     ('$'+HOUR+MINUTE+'_'+MONTH+DAY+'_'+SEQUENCE)
  ```

- Setting the NO BATCH UPDATE or NO EXCLUSIVE transaction modes prevents various transaction types on IMPORT and can effectively prevent the import from succeeding.
- Oracle Rdb prevents user specification of the disabled transactions modes when the transaction parameter block (TPB) is processed.

## Examples

Example 1: Changing a read/write storage area to a read-only storage area

This example changes the SALARY_HISTORY storage area from a read/write storage area to a read-only storage area.

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ALTER STORAGE AREA salary_history
cont> READ ONLY;
```

Example 2: Changing a read/write storage area to a write-once area

This example changes the RESUME_LISTS storage area from a read/write storage area to a write-once storage area.

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ALTER STORAGE AREA resume_lists
cont> WRITE ONCE;
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SHOW STORAGE AREA resume_lists;

     RESUME_LISTS
         Page Format:    Mixed
         Page Size:      6 blocks
         .
         .
         .
         Locking is Row Level
  Write Once (Journal is Enabled)

Database objects using Storage Area RESUME_LISTS:
Usage           Object Name                      Map / Partition
--------------- -------------------------------- -----------------------------
List Storage Map                                 LISTS_MAP (1)
```

Example 3: Disabling extents

This example demonstrates disabling extents using the ALTER DATABASE statement. Because extents can be altered on line, you can be attached to the database while you alter extents.

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SHOW STORAGE AREA EMPIDS_LOW
```

```
     EMPIDS_LOW
         Access is:      Read write
         Page Format:    Mixed
         Page Size:      2 blocks
         .
         .
         .
         Snapshot Allocation:      10 pages
         Snapshot Extent Minimum:  99 pages
         Snapshot Extent Maximum:  9999 pages
         Snapshot Extent Percent:  20 percent
         Extent :       Enabled
         Locking is Row Level
         .
         .
         .
SQL> DISCONNECT ALL;
SQL> ALTER DATABASE FILENAME mf_personnel
cont> ALTER STORAGE AREA EMPIDS_LOW
cont> EXTENT DISABLED;
SQL> ATTACH 'FILENAME mf_personnel;
SQL> SHOW STORAGE AREA EMPIDS_LOW
     EMPIDS_LOW
         Access is:      Read write
         Page Format:    Mixed
         Page Size:      2 blocks
         .
         .
         .
         Snapshot Allocation:      10 pages
         Snapshot Extent Minimum:  99 pages
         Snapshot Extent Maximum:  9999 pages
         Snapshot Extent Percent:  20 percent
         Extent :       Disabled
         Locking is Row Level
         .
         .
         .
```

Example 4: Adding multiple, fixed-size journal files

This example demonstrates reserving slots for journal files, enabling the journaling feature, and adding multiple, fixed-size journal files.

```
SQL> CREATE DATABASE FILENAME test
cont>    RESERVE 5 JOURNALS
cont>    CREATE STORAGE AREA sa_one
cont>       ALLOCATION IS 10 PAGES;
SQL> DISCONNECT ALL;
SQL>
SQL> ALTER DATABASE FILENAME test
cont>    JOURNAL IS ENABLED
cont>    ADD JOURNAL AIJ_ONE
cont>       FILENAME aij_one
cont>       BACKUP FILENAME aij_one
cont>    ADD JOURNAL AIJ_TWO
cont>       FILENAME aij_two
cont>       BACKUP FILENAME aij_two
cont> ;
```

You should place journal files and backup files on disks other than those that contain the database.

Example 5: Reserving and using slots for storage areas

This example demonstrates reserving slots for storage areas and adding storage areas to the database that utilizes those slots. Use the SHOW DATABASE statement to see changes made to the database.

```
SQL> CREATE DATABASE FILENAME sample
cont>    RESERVE 5 STORAGE AREAS
cont>    CREATE STORAGE AREA RDB$SYSTEM
cont>       FILENAME sample_system
cont> --
cont> -- Storage areas created when the database is created do not use
cont> -- the reserved storage area slots because this operation is being
cont> -- executed off line.
cont> --
cont> ;
%RDMS-W-DOFULLBCK, full database backup should be done to ensure future
recovery
SQL> --
SQL> -- Reserving storage area slots is not a journaled activity.
SQL> --
SQL> -- To use the reserved slots, you must alter the database and
SQL> -- add storage areas.
SQL> --
SQL> DISCONNECT ALL;
SQL> ALTER DATABASE FILENAME sample
cont>    ADD STORAGE AREA SAMPLE_1
cont>       FILENAME sample_1
cont>    ADD STORAGE AREA SAMPLE_2
cont>       FILENAME sample_2;
```

## ALTER DOMAIN Statement

Alters a domain definition.

A **domain** is the set of values that a column in a table can have. A domain definition specifies the set of values by associating an SQL data type with a domain name. The CREATE and ALTER TABLE statements can use the domain in column definitions.

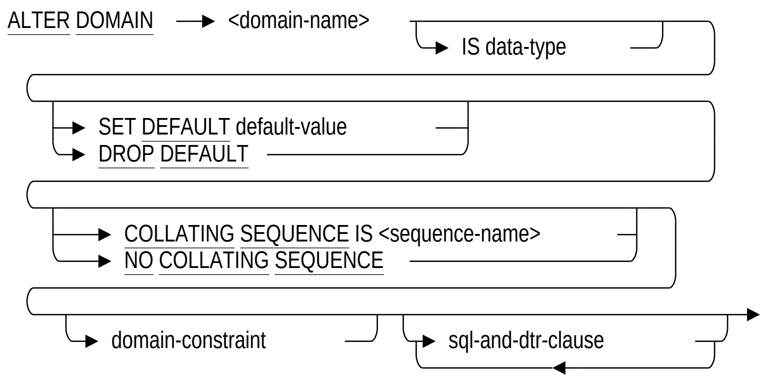
The ALTER DOMAIN statement lets you change the character set, data type, optional default value, optional collating sequence, or optional formatting and DATATRIEVE clauses associated with a domain name. Any table definitions that refer to that domain reflect the changes.

### Environment

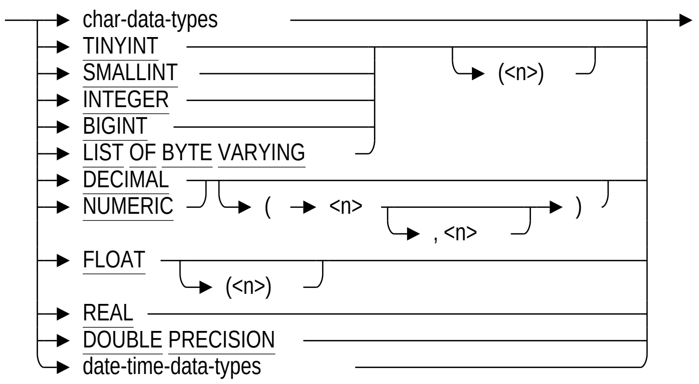
You can use the ALTER DOMAIN statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

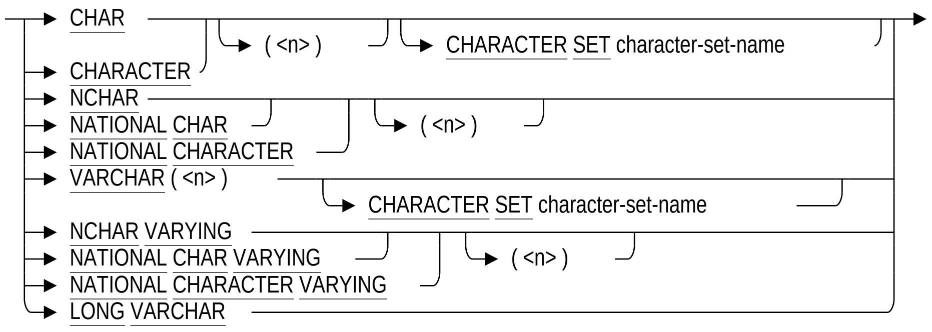
### Format



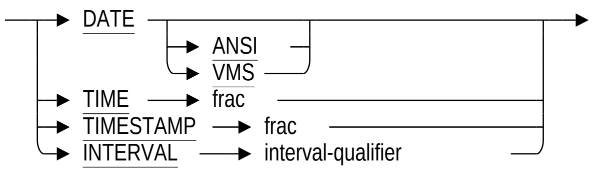
data-type =



char-data-types =

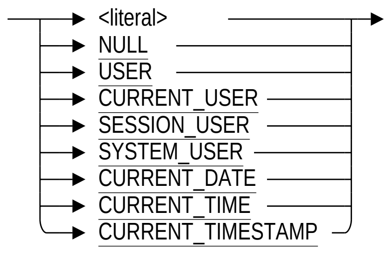


date-time-data-types =

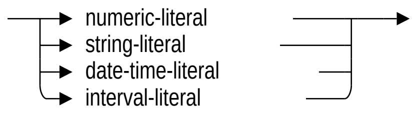


## ALTER DOMAIN Statement

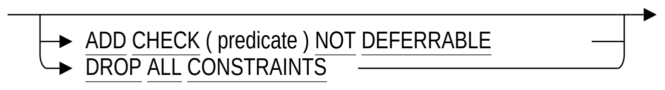
default-value =



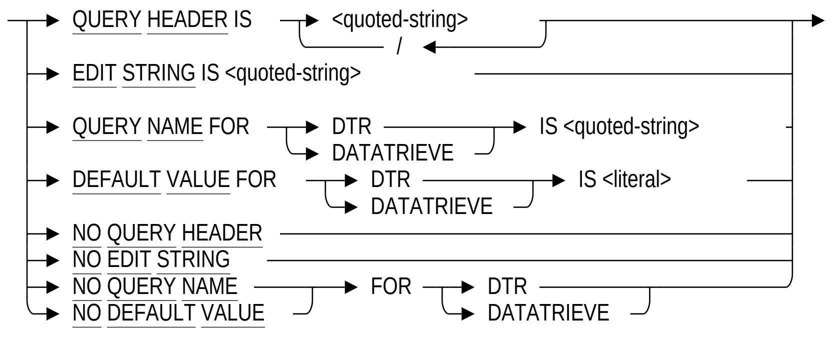
literal =



domain-constraint =



sql-and-dtr-clause =



## Arguments

**domain-name**

The name of a domain you want to alter. The domain name must be unique among domain names in the database.

**IS data-type**
A valid SQL data type. For more information on data types, see Section 2.3.

**char-data-types**
A valid SQL character data type. For more information on character data types, see Section 2.3.1.

**character-set-name**
A valid character set name. For a list of allowable character set names, see Section 2.1.

**date-time-data-types**
A data type that specifies a date, time, or interval. For more information on date-time data types, see Section 2.3.5.

**SET DEFAULT**
Provides a default value for a column if the row that is inserted does not include a value for that column. A column default value overrides a domain default value. You can use literals, the NULL keyword, the user name, the session user name, the system user name, the current date, the current time, or the current timestamp as default values. If you do not specify a default value, SQL assigns NULL as the default value. For more information about NULL, see Section 2.6.1 and the Usage Notes following this Arguments list.

**default-value**
Specifies the default value of a domain. The following table lists the valid values:

| Default Value | Description |
|---|---|
| literal | A value expression. Literal values can be numeric, character string, or date data types. |
| NULL | A null value. |
| USER | The current, active user name for a request. |
| CURRENT_USER | The current, active user name for a request. If a definer's rights request is executing, SQL returns the definer's user name. If not, SQL returns the session user name, if it exists. Otherwise, SQL returns the system user name. |

| Default Value | Description |
|---|---|
| SESSION_USER | The current, active session user name. If the session user name does not exist, SQL returns the system user name. |
| SYSTEM_USER | The user name of the process at the time of the database attach. |
| CURRENT_DATE | The DATE data type value containing year, month, and day for date “today”. |
| CURRENT_TIME | The TIME data type value containing hours, minutes, and seconds for time “now”. |
| CURRENT_ TIMESTAMP | The date and time currently defined in Oracle Rdb. |

**DROP DEFAULT**
Deletes (drops) the default value of a domain.

OpenVMS VAX OpenVMS Alpha

**COLLATING SEQUENCE IS sequence-name**
Specifies a new collating sequence for the named domain.

The OpenVMS National Character Set (NCS) utility provides a set of predefined collating sequences and also lets you define collating sequences of your own. The COLLATING SEQUENCE clause accepts both predefined and user-defined NCS collating sequences.

Before you use the COLLATING SEQUENCE clause in an ALTER DOMAIN statement, you must first specify the NCS collating sequence for SQL using the CREATE COLLATING SEQUENCE statement. The sequence name argument in the COLLATING SEQUENCE clause must be the same as the sequence name in the CREATE COLLATING SEQUENCE statement. ♦

OpenVMS VAX OpenVMS Alpha

**NO COLLATING SEQUENCE**
Specifies that the named domain uses the standard default collating sequence, that is, ASCII. Use the NO COLLATING SEQUENCE clause to override the collating sequence defined for the schema in the CREATE SCHEMA or ALTER SCHEMA statement, or the domain in the CREATE DOMAIN statement. ♦

**domain-constraint**
Adds or modifies a constraint for the existing named domain.

**Domain constraints** specify that columns based on the domain contain only certain data values or that data values can or cannot be null.

Use the CHECK clause to specify that a value must be within a specified range or that it matches a list of values. When you specify a CHECK clause for a domain constraint, you ensure that all values stored in columns based on the domain are checked consistently.

To refer to the values of all columns of a domain constraint, use the VALUE keyword. For example:

```
SQL> CREATE DOMAIN dom1 CHAR(1)
cont> CHECK (VALUE IN ('F','M'))
cont> NOT DEFERRABLE;
```

For any dialect other than SQL92, you must specify that domain constraints are NOT DEFERRABLE.

When you add (or modify) a domain constraint, SQL propagates the new constraint definition to all the columns that are based on the domain. If columns that are based on the domain contain data that does not conform to the constraint, SQL returns the following error:

```
%RDB-E-NOT_VALID, validation on field DATE_COL caused operation to fail
```

To modify a domain constraint, you must first delete any existing domain constraint using the DROP ALL CONSTRAINTS clause of the ALTER DOMAIN statement. Then, you add the new domain constraint using the ADD constraint clause of the ALTER DOMAIN statement.

A column default value overrides the domain default value.

**sql-and-dtr-clause**
Optional SQL and DATATRIEVE formatting clause. For more information on the formatting clauses, see Section 2.5.

## Usage Notes

- You cannot alter a domain definition unless you have declared the database that includes the domain.
- Because Oracle Rdb creates dependencies between stored procedures and metadata (like domains) on which they are compiled and stored, you cannot alter a domain if the domain is used in a parameter list of a stored procedure. However, you can alter a domain if that domain is referenced within the procedure block. See the example in this section about creating stored procedure domain dependencies and the effect this has on the ALTER DOMAIN statement.

- The ALTER DOMAIN statement lets you change the data type, optional default value, optional collating sequence, or optional formatting and DATATRIEVE clauses for all columns defined using the domain by changing the domain itself. For example, if you want to change the data type for EMPLOYEE_ID from CHAR(5) to CHAR(6), you need only alter the data type for ID_DOM. You do not have to alter the data type for the column EMPLOYEE_ID in the tables DEGREES, EMPLOYEES, JOB_HISTORY, or SALARY_HISTORY, nor do you have to alter the column MANAGER_ID in the DEPARTMENTS table. (However, if the EMPLOYEE_ID domain is referred to in an index or view definition, see the next note.)
- You cannot issue an ALTER DOMAIN statement changing the data type or collating sequence of a domain that is referred to in an index definition. To change the data type or collating sequence in such cases, you must first delete the index, change the domain, then define the index again.
- The data type of a value specified in the DEFAULT VALUE clause must be the same data type as the column in which it is defined. If you forget to specify the data type, SQL issues an error message, as shown in the following example:

  ```
  SQL> CREATE DOMAIN TIME_DOM IS TIME ( 2 ) DEFAULT '00:00:00.00' ;
  %SQL-F-DEFVALINC, You specified a default value for TIME_DOM which is
  inconsistent with its data type
  SQL> CREATE DOMAIN TIME_DOM IS TIME ( 2 ) DEFAULT TIME '00:00:00.00' ;
  ```
- The result data type for the USER, CURRENT_USER, SESSION_USER, and SYSTEM_USER keywords is CHAR(31).
- The ALTER DOMAIN statement allows you to change the character set associated with a domain name. However, if this is done after data is entered into a table using the domain name, SQL returns a data conversion error when you try to select rows from that table.
- You can specify the national character data type by using the NCHAR, NATIONAL CHAR, NCHAR VARYING, or NATIONAL CHAR VARYING data types. The national character data type is defined by the database national character set when the database is created. For more information on national character data types, see Section 2.3.
- You can specify the length of the data type in characters or octets. By default, data types are specified in octets. By preceding the ALTER DOMAIN statement with the SET CHARACTER LENGTH 'CHARACTERS' or SET DIALECT 'MIA' or SET DIALECT 'SQL92' statement, you change the length to characters. For more information,

see the SET CHARACTER LENGTH Statement and the SET DIALECT Statement.

- You should consider what value, if any, you want to use for the default value for a domain. You can use a value such as NULL or "Not Applicable" that clearly demonstrates that no data was inserted into a column based on that domain. If a column usually contains a particular value, you could use that value as the default. For example, if most company employees live in the same state, you could make that state the default value for the STATE_DOM column.

  A default value specified for a column overrides a default value specified for the domain.

  To remove a default value, use the DROP DEFAULT clause.

  If you change or add a default value for a domain, the change has no effect on any existing data in the database; that is, the rows already stored in the database with columns that contain the old default value are not changed.

  The default value is not the same as the missing value that you can specify using the RDO interface. In contrast to default values, changing the missing value does change what is displayed by applications based on RDO for columns that have no data value stored and that have a missing value defined. For a discussion of the difference between default value and missing value, see the *Oracle Rdb7 Guide to Database Design and Definition*.

- Changes you make to domains created with the FROM clause (based on a repository definition) can affect other applications. If the database was declared with the PATHNAME clause, changes made with the ALTER DOMAIN statement are immediately written to the repository record or field definitions. If the database was declared with the FILENAME clause, the changes are written to the repository when the next INTEGRATE SCHEMA . . . ALTER DICTIONARY statement is issued.

  The changes affect applications and other databases that use the same repository definition when the application recompiles or the database integrates with the repository.

  For this reason, use caution when you alter domains that are based on repository definitions. Make sure that changes you make through ALTER DOMAIN statements do not have unintended effects on other users or applications that share the repository definitions.

- You must execute the ALTER DOMAIN statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with characteristics specified in the most recent DECLARE TRANSACTION statement.
- You cannot execute the ALTER DOMAIN statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. For more information on the RDB$SYSTEM storage area, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.
- The ALTER DOMAIN statement fails when both of the following circumstances are true:
  - The database to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The database was declared using the FILENAME argument.

  Under these circumstances, the statement fails with the following error when you issue it:

  ```
  %RDB-E-NO_META_UPDATE, metadata update failed
  -RDMS-F-CDDISREQ, CDD required for metadata updates
       is not being maintained
  ```
- Suppose you perform an ALTER DOMAIN operation that causes a conversion error on retrieval of a record. In an attempt to avoid the error, you might try to delete the record. This will not work because the delete operation attempts to do the same incorrect conversion.

  A workaround to this problem is to alter or change the domain back to the original data type, and then remove or change the offending records. Then, you can use the ALTER DOMAIN statement to alter the domain to the new required data type.
- When adding a domain constraint, the predicate cannot contain subqueries and cannot refer to another domain.
- You can only specify one constraint for each domain.
- The CHECK constraint syntax can reference the VALUE keyword or the domain name. For example:

```
SQL> -- The CHECK constraint can reference the VALUE keyword.
SQL> --
SQL> ALTER DOMAIN D1 INTEGER
cont> ADD CHECK (VALUE > 10)
cont> NOT DEFERRABLE;
SQL> SHOW DOMAIN D1;
D1                             INTEGER
 Valid If:       (VALUE > 10)
SQL> ROLLBACK;
SQL> --
SQL> -- The CHECK constraint can reference the domain name.
SQL> --
SQL> ALTER DOMAIN D1 INTEGER
cont> ADD CHECK (D1 > 10)
cont> NOT DEFERRABLE;
SQL> SHOW DOMAIN D1
D1                             INTEGER
 Valid If:       (D1 > 10)
```

- You can alter the data type of a domain with a referencing NOT NULL constraint without first deleting the constraint.
- You can change the data type of a domain that is referenced by a column used in a trigger definition and possibly invalidate the trigger. Existing data may violate the trigger after the data type change. Before altering a domain that is referenced by a column in a trigger definition, verify that the new data type is consistent with the previously defined trigger.
- Because of some special characteristics of the Norwegian collating sequence, certain restrictions apply when creating a Norwegian collating sequence in a database. The name of a Norwegian collating sequence in the NCS library must begin with the character string NORWEGIAN.

  The sequence customarily shipped with OpenVMS is named NORWEGIAN which meets this restriction. You may wish to alter the Norwegian sequence slightly or change its name. Oracle recommends that any variation of the Norwegian collating sequence be given a name such as NORWEGIAN_1 or NORWEGIANA.
- You cannot alter a domain that is referenced by a temporary table once data has been inserted into the temporary table.

## Examples

Example 1: Altering the domain POSTAL_CODE_DOM

This example alters the domain POSTAL_CODE_DOM so that it accommodates longer postal codes:

```
SQL> --
SQL> -- The data type of the current domain POSTAL_CODE_DOM is CHAR(5):
SQL> --
SQL> SHOW DOMAIN POSTAL_CODE_DOM
POSTAL_CODE_DOM                 CHAR(5)
 Comment:       standard definition of ZIP
 Rdb default:
SQL> --
SQL> -- Now, alter the domain to accommodate larger postal codes:
SQL> --
SQL> ALTER DOMAIN POSTAL_CODE_DOM IS CHAR(10);
SQL> --
SQL> -- The SHOW TABLES statement shows how changing the
SQL> -- domain POSTAL_CODE_DOM changes all the
SQL> -- columns that were created using the domain:
SQL> --
SQL> SHOW TABLE COLLEGES
Information for table COLLEGES

Comment on table COLLEGES:
names and addresses of colleges attended by employees

Columns for table COLLEGES:
Column Name                     Data Type       Domain
-----------                     ---------       ------
.
.
.
POSTAL_CODE                     CHAR(10)        POSTAL_CODE_DOM
.
.
.

SQL> SHOW TABLE EMPLOYEES
Information for table EMPLOYEES

Comment on table EMPLOYEES:
personal information about each employee
```

```
Columns for table EMPLOYEES:
Column Name                     Data Type        Domain
-----------                     ---------        ------
.
.
.
POSTAL_CODE                     CHAR(10)         POSTAL_CODE_DOM
```

Example 2: Altering the domain ID_DOM

The following example alters the data type for the domain ID_DOM, which is a standard definition of the employee identification field.

In Example 1, there were no indexes based on the domain POSTAL_CODE_DOM. In this example, several indexes that refer to columns were created based on ID_DOM. As the following example shows, you must first delete the indexes before altering the domain:

```
SQL> -- The data type for the domain ID_DOM is CHAR(5):
SQL> --
SQL> SHOW DOMAIN ID_DOM
ID_DOM                          CHAR(5)
 Comment:       standard definition of employee id
SQL> --
SQL> -- The first attempt to alter the domain ID_DOM fails.
SQL> -- You must first delete all constraints that use the
SQL> -- field EMPLOYEE_ID.
SQL> --
SQL> ALTER DOMAIN ID_DOM CHAR(6);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-FLDINCON, field EMPLOYEE_ID is referenced in constraint
RESUMES_FOREIGN1
-RDMS-F-FLDNOTCHG, field EMPLOYEE_ID has not been changed
SQL> ALTER TABLE RESUMES DROP CONSTRAINT RESUMES_FOREIGN1;
SQL> --
SQL> ALTER DOMAIN ID_DOM IS CHAR(6);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-FLDINCON, field EMPLOYEE_ID is referenced in constraint
DEGREES_FOREIGN1
-RDMS-F-FLDNOTCHG, field EMPLOYEE_ID has not been changed
SQL> --
SQL> ALTER TABLE DEGREES DROP CONSTRAINT DEGREES_FOREIGN1;
   .
   .
   .
SQL> -- You must then delete all indexes.
SQL> --
SQL> ALTER DOMAIN ID_DOM IS CHAR(6);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-FLDINUSE, field EMPLOYEE_ID is referenced in index EMP_EMPLOYEE_ID
-RDMS-F-FLDNOTCHG, field EMPLOYEE_ID has not been changed
SQL> --
SQL> DROP INDEX EMP_EMPLOYEE_ID;
SQL> --
SQL> ALTER DOMAIN ID_DOM IS CHAR(6);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-FLDINUSE, field EMPLOYEE_ID is referenced in index JH_EMPLOYEE_ID
-RDMS-F-FLDNOTCHG, field EMPLOYEE_ID has not been changed
SQL> --
SQL> DROP INDEX JH_EMPLOYEE_ID;
SQL> --
   .
   .
   .
SQL> --
SQL> -- You can now alter the domain.
SQL> --
SQL> ALTER DOMAIN ID_DOM IS CHAR(6);
SQL> SHOW DOMAIN ID_DOM;
ID_DOM                          CHAR(6)
 Comment:       standard definition of employee id
```

Example 3: Specifying default values with the ALTER DOMAIN statement

The following example alters domains, specifying default values for those domains:

```
SQL> -- If no date is entered, use the NULL default.
SQL> --
SQL> ALTER DOMAIN DATE_DOM
cont> SET DEFAULT NULL;
SQL> --
SQL> -- If the street address takes only one line,
SQL> -- use "NONE" for the default for the second line.
SQL> --
SQL> ALTER DOMAIN ADDRESS_DATA_2_DOM
cont> SET DEFAULT 'NONE';
SQL> --
SQL> -- If most employees work full-time, make the code
SQL> -- for full-time, 1, the default work status.
SQL> --
SQL> ALTER DOMAIN STATUS_CODE_DOM
cont> SET DEFAULT '1';
```

Example 4: Specifying an edit string with the ALTER DOMAIN statement

The following example specifies an EDIT STRING clause that controls how SQL displays columns based on the domain MIDDLE_INITIAL_DOM. The edit string in the example, "X.?'No middle initial'", specifies that columns based on the domain are displayed as one character followed by a period. If there is no value for the column, SQL displays the string following the question mark, 'No middle initial'.

```
SQL> ALTER DOMAIN MIDDLE_INITIAL_DOM
cont>   EDIT STRING 'X.?''No middle initial';
SQL> SELECT MIDDLE_INITIAL FROM EMPLOYEES;
 MIDDLE_INITIAL
 A.
 D.
 No middle initial
 No middle initial
        .
        .
        .
```

Example 5: Specifying a new collating sequence with the ALTER DOMAIN statement

The following example creates a domain with the predefined NCS collating sequence FRENCH. You must first execute the CREATE COLLATING SEQUENCE statement. The example then changes the collating sequence to Finnish, and then specifies that the domain has no collating sequence.

```
SQL> CREATE COLLATING SEQUENCE FRENCH FRENCH;
SQL> CREATE DOMAIN LAST_NAME_ALTER_TEST CHAR (10)-
cont> COLLATING SEQUENCE IS FRENCH;
SQL> --
SQL> SHOW DOMAIN LAST_NAME_ALTER_TEST
LAST_NAME_ALTER_TEST            CHAR(10)
 Collating sequence: FRENCH
SQL> --
SQL> -- Now, change the collating sequence to Finnish.  You must first
SQL> -- execute the CREATE COLLATING SEQUENCE statement.
SQL> --
SQL> CREATE COLLATING SEQUENCE FINNISH FINNISH;
SQL> ALTER DOMAIN LAST_NAME_ALTER_TEST CHAR (10)-
cont> COLLATING SEQUENCE IS FINNISH;
SQL> --
SQL> SHOW DOMAIN LAST_NAME_ALTER_TEST
LAST_NAME_ALTER_TEST            CHAR(10)
 Collating sequence: FINNISH
SQL> --
SQL> -- Now, alter the domain so there is no collating sequence.
SQL> --
SQL> ALTER DOMAIN LAST_NAME_ALTER_TEST CHAR (10)-
cont> NO COLLATING SEQUENCE;
SQL>
SQL> SHOW DOMAIN LAST_NAME_ALTER_TEST
LAST_NAME_ALTER_TEST            CHAR(10)
```

Assume the following for Examples 6 and 7:

- The database was created specifying the database default character set as DEC_KANJI and the national character set as KANJI.
- The domain DEC_KANJI_DOM was created specifying the database default character set.
- The table COLOURS was created assigning the DEC_KANJI_DOM domain to the column ROMAJI.

Example 6: Altering the domain DEC_KANJI_DOM

```
SQL> SET CHARACTER LENGTH 'CHARACTERS';
SQL> SHOW DOMAIN DEC_KANJI_DOM;
DEC_KANJI_DOM                   CHAR(8)
SQL> ALTER DOMAIN DEC_KANJI_DOM NCHAR(8);
SQL> SHOW DOMAIN DEC_KANJI_DOM;
DEC_KANJI_DOM                   CHAR(8)
         KANJI 8 Characters,  16 Octets
SQL>
```

Example 7: Error altering a domain used in a table containing data

In the following example, the column ROMAJI is based on the domain DEC_KANJI_DOM. If the column ROMAJI contains data before you alter the character set of the domain, SQL displays the following error when you try to retrieve data after altering the domain.

```
SQL> SELECT ROMAJI FROM COLOURS;
%RDB-F-CONVERT_ERROR, invalid or unsupported data conversion
-RDMS-E-CSETBADASSIGN, incompatible character sets prohibits the requested
 assignment
SQL> --
SQL> -- To recover, use the ROLLBACK statement or reset the character set to
SQL> -- its original value.
SQL> --
SQL>ROLLBACK;
SQL> SELECT ROMAJI FROM COLOURS;
 ROMAJI
 kuro
 shiro
 ao
 aka
 ki
 midori
6 rows selected
SQL>
```

Example 8: Modifying a domain constraint

The following example shows how to modify an existing constraint on a domain:

```
SQL> SHOW DOMAIN TEST_DOM
TEST_DOM                        DATE ANSI
 Rdb default: NULL
 CHECK:  (VALUE > DATE’1900-01-01’ OR
                             VALUE IS NULL)
SQL> --
SQL> -- You must delete the constraint before you can modify it.
SQL> --
SQL> ALTER DOMAIN TEST_DOM
cont>   DROP ALL CONSTRAINTS;
SQL> SHOW DOMAIN TEST_DOM
TEST_DOM                        DATE ANSI
 Rdb default: NULL
SQL> --
SQL> -- Add the new domain constraint definition.
SQL> --
SQL> ALTER DOMAIN TEST_DOM
cont>   ADD CHECK (VALUE > DATE’1985-01-01’)
cont>   NOT DEFERRABLE;
```

Example 9: Creating stored procedure domain dependencies

The following code fragment from a stored module shows a domain in a parameter list and a domain in a stored procedure block:

```
PROCEDURE domain_p (:in_var id_number)
COMMENT IS ’This procedure creates domain dependencies’;
BEGIN
    declare :local_var last_name;
    insert into employees (middle_initial)
        values (cast (’1’ as middle_initial));
END;
```

- Domain specified in a parameter list

  When you specify a domain in a parameter list (id_number) of a stored routine and you subsequently try to alter that domain, the ALTER DOMAIN statement fails because SQL sets up a dependency between the domain and the stored routine in which the domain resides. Because the statement fails, Oracle Rdb does not invalidate the stored routine. Oracle Rdb keeps this domain parameter list dependency in RDB$PARAMETERS, not in RDB$INTERRELATIONS where it usually keeps dependency information.

- Domain specified in a stored routine block

  When you specify a domain (last_name) within a stored routine block and you subsequently try to alter that domain, the ALTER DOMAIN statement succeeds, but the operation does not mark the stored routine invalid. Oracle Rdb keeps this domain stored routine block dependency in RDB$INTERRELATIONS where it usually keeps dependency information.

## ALTER INDEX Statement

Changes an index. The ALTER INDEX statement allows you to change the:

- Characteristics of index nodes (sorted indexes only)
- Names of the storage areas that contain the index

You cannot change:

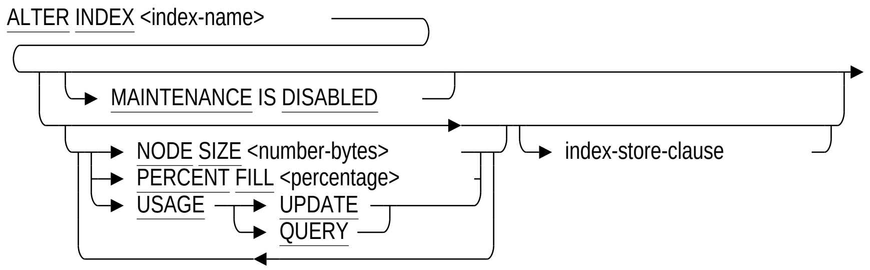
- The columns that comprise an index
- Whether or not the index is UNIQUE
- A hashed index to a sorted index
- A sorted index to a hashed index
- A sorted, nonranked index to a sorted, ranked index
- A sorted, ranked index to a sorted, nonranked index
- The duplicates compression of a sorted, ranked index

### Environment

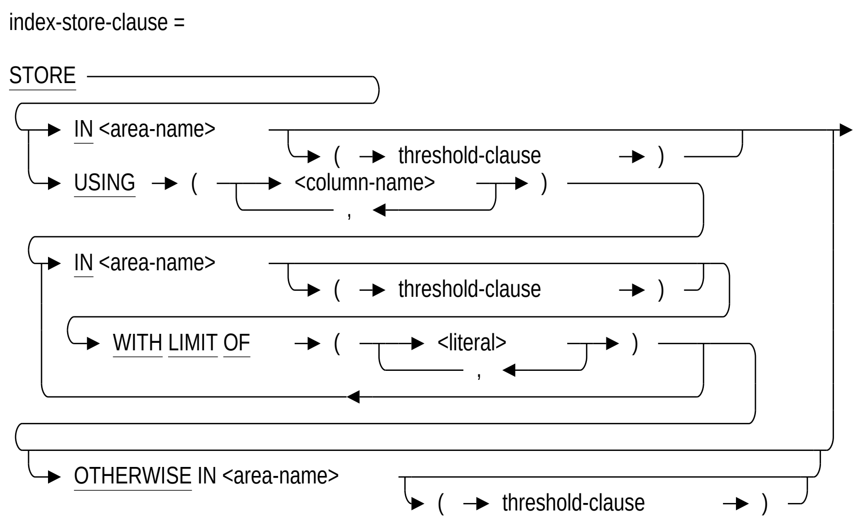
You can use the ALTER INDEX statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

### Format



## ALTER INDEX Statement



## Arguments

**index-name**
The name of the index.

**MAINTENANCE IS DISABLED**
Disables, but does not delete, the specified index.

When managing a very large database, an index can become corrupt or unsuitable for query optimization. If the table on which the index has been defined is very large, it may take a considerable amount of time to execute the DROP INDEX statement. Using the MAINTENANCE IS DISABLED clause of the ALTER INDEX statement permanently disables the index so that it is no longer used by the optimizer nor is it maintained. You can then execute the DROP INDEX statement at a later time when the database table can be taken off line.

Once an index has been disabled, it cannot be enabled again.

To disable an index, you must have delete privileges to the table on which the index is defined, and there can be no active queries on the table.

**NODE SIZE number-bytes**
The size, in bytes, of each index node in a sorted index. You cannot specify this argument in an ALTER INDEX statement that refers to a hashed index. See the CREATE INDEX Statement for details of the NODE SIZE clause.

**PERCENT FILL percentage**
Specifies how much each index node should be filled as a percentage of its size. You cannot specify this argument in an ALTER INDEX statement that refers to a hashed index. The valid range is 1 percent to 100 percent. The default is 70 percent.

Both the PERCENT FILL and USAGE clauses specify how full an index node should be initially. You should specify either the PERCENT FILL or USAGE clause but not both. However, if you do, SQL uses the last clause specified.

**USAGE UPDATE**
**USAGE QUERY**
Specifies a PERCENT FILL value appropriate for update-intensive or query-intensive applications. You cannot specify this argument in an ALTER INDEX statement that refers to a hashed index. The USAGE UPDATE argument sets the PERCENT FILL value at 70 percent. The USAGE QUERY argument sets the PERCENT FILL value at 100 percent.

Supplying both the PERCENT FILL and USAGE clauses is allowed in the syntax but is semantically meaningless. You should specify either the PERCENT FILL or USAGE clause but not both. However, if you specify both, SQL uses the last clause specified.

**index-store-clause**
A storage map definition for the index. You can specify a store clause for indexes in a multifile database only. The STORE clause lets you specify which storage area files are used to store the index entries.

If you omit the storage map definition, the default is to store all entries for the index in the main RDB$SYSTEM storage area.

See the CREATE INDEX Statement for details of the arguments in an index store clause.

## Usage Notes

- You must execute the ALTER INDEX statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with characteristics specified in the most recent DECLARE TRANSACTION statement.

- Attempts to alter an index will fail if that index is involved in a query at the same time. Users must detach from the database with a DISCONNECT statement before you can alter the index. When Oracle Rdb first accesses an object, such as the index, a lock is taken out on that object and not released until the user exits the database. If you attempt to update this object, you will receive a LOCK CONFLICT ON CLIENT message due to the other user's access to the object.

  Similarly, while you alter an index, users cannot execute queries involving that index until you completed the transaction with a COMMIT or ROLLBACK statement for the ALTER statement; otherwise the users receive a LOCK CONFLICT ON CLIENT error message. While data definition language (DDL) operations are performed, normal data locking mechanisms are used against system tables. (System tables contain information about objects in the database.) Therefore, attempts to update an object locks out attempts to query that object. These locks are held until the DDL operation is committed or rolled back.

  The WAIT/NOWAIT clause of the SET TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify SET TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

  ```
  %RDB-E-LOCK_CONFLICT, request failed due to locked resource;
  no-wait parameter specified for transaction
  -RDB-E-NO_META_UPDATE, metadata update failed
  -RDMS-F-LCKCNFLCT, lock conflict on client
  SQL>
  ```

  However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement even if the user specified NOWAIT on the SET TRANSACTION statement.

- You cannot alter compression clauses for index columns using the SIZE IS and MAPPING VALUES clauses. You must delete the index and re-create it to alter such clauses.

- The ALTER INDEX statement fails when both of the following circumstances are true:

  - The schema to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The schema was declared using the FILENAME argument.

Under these circumstances, the statement fails with the following error when you issue it:

```
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-CDDISREQ, CDD required for metadata updates
                  is not being maintained
```

- You cannot execute the ALTER INDEX statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. For more information on the RDB$SYSTEM storage area, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

## Examples

Example 1: Disabling an index

The following example shows how to disable an index that can be deleted at a later time when the database table can be taken off line:

```
SQL> ALTER INDEX MY_NDX
cont> MAINTENANCE IS DISABLED;
SQL> SHOW INDEX MY_NDX;
Indexes on table EMPLOYEES:
MY_NDX                          with column EMPLOYEE_ID
  Duplicates are allowed
  Type is Sorted
  Compression is ENABLED  (Minimum run length  2)
  Index is no longer maintained
```

## ALTER STORAGE MAP Statement

Changes an existing storage map. A storage map controls which rows of a table are stored in which storage areas in a multifile database.

In addition to changing storage maps, the ALTER STORAGE MAP statement has options that change the following:

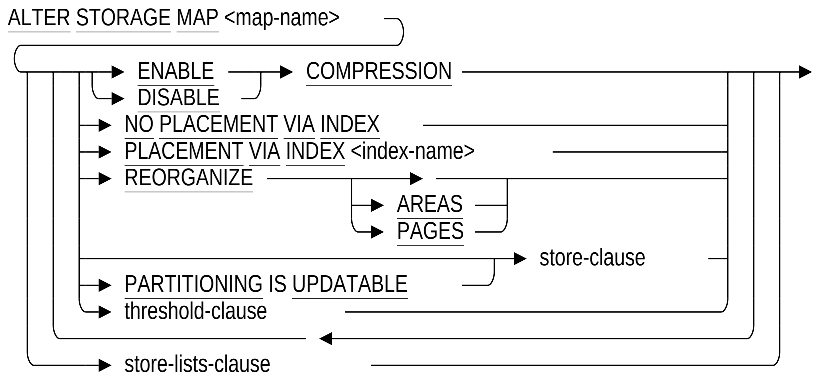
- Which index the database system uses when inserting rows in the table
- Whether or not the rows of the table are stored in a compressed format
- Whether or not the data is reorganized
- Whether partitioning keys can be modified

### Environment

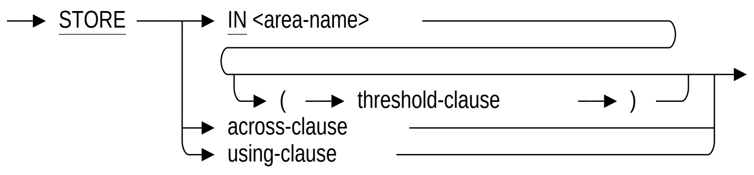
You can use the ALTER STORAGE MAP statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

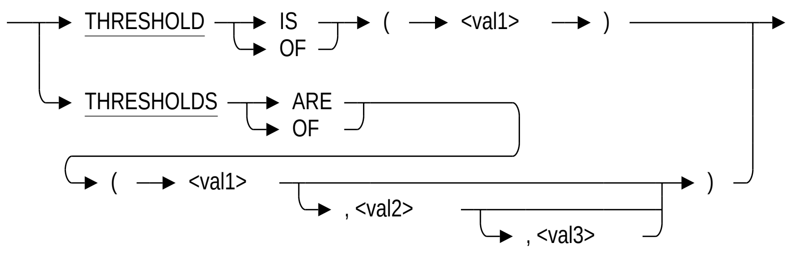
### Format



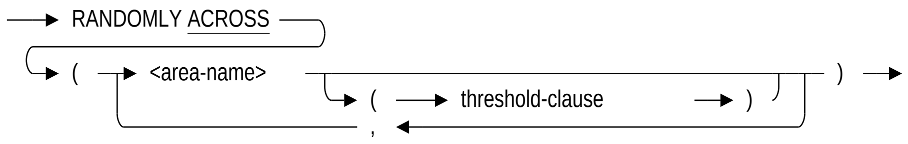
store-clause =



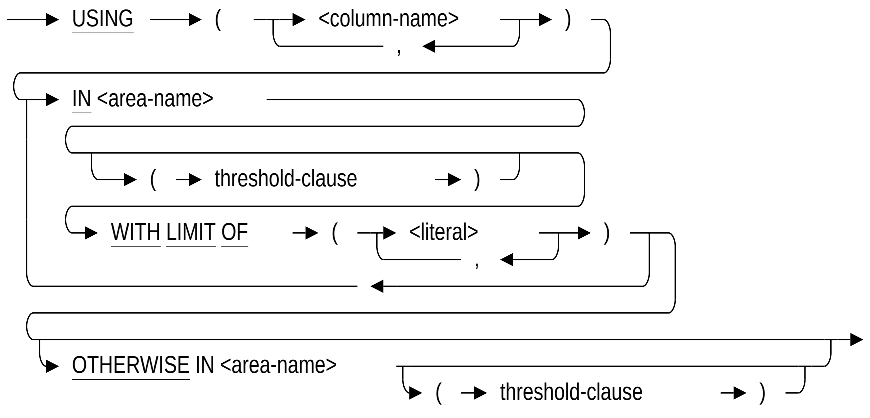
threshold-clause =



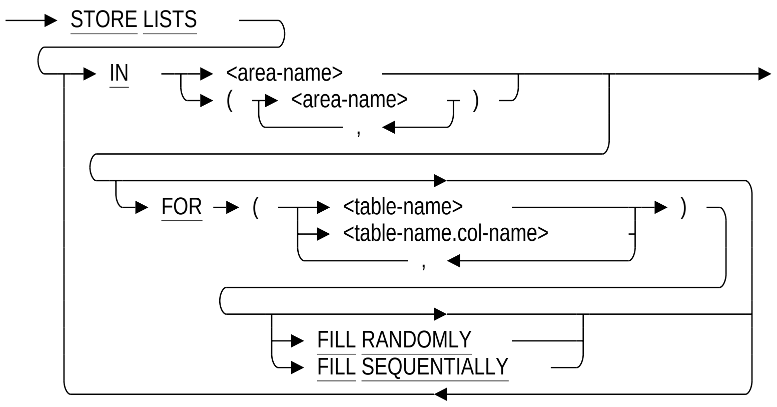
across clause =



using-clause =



store-lists-clause =



## Arguments

**STORAGE MAP map-name**
Specifies the name of the storage map you want to alter.

**ENABLE COMPRESSION**
**DISABLE**
Changes whether the rows for the table are compressed or uncompressed when stored. Enabling compression conserves disk space, but it incurs additional CPU overhead for inserting and retrieving compressed rows.

Changing the COMPRESSION clause causes the database system to read all the rows in the table and write them back to the table in the changed format. If compression is enabled and you subsequently disable it, records may become fragmented because the space allowed for the record is no longer large enough.

**NO PLACEMENT VIA INDEX**
Negates the PLACEMENT VIA INDEX clause so that subsequent records stored are not stored by means of the index named in the PLACEMENT VIA INDEX clause. This argument is available only for the ALTER STORAGE MAP statement. If you specify the ALTER STORAGE MAP statement without the PLACEMENT VIA INDEX argument or the NO PLACEMENT VIA INDEX argument, the statement executes as if the clause specified on the CREATE STORAGE MAP statement or last ALTER STORAGE MAP statement was used.

**PLACEMENT VIA INDEX index-name**
See the CREATE STORAGE MAP Statement for details of the PLACEMENT VIA INDEX argument.

**Note**

You can define a single PLACEMENT VIA INDEX clause to place the primary partitioning keys. You must specify it on the first vertical partition. You can, optionally, specify the NO PLACEMENT VIA INDEX clause on the first partition. However, on subsequent partitions you can specify only the NO PLACEMENT VIA INDEX clause.

**REORGANIZE**
Causes new rows and rows previously stored in specified tables to be moved according to the partitions specified in the STORE clause of the ALTER STORAGE MAP statement. The REORGANIZE clause works for one or more areas in the storage maps.

For details of how rows are moved or not moved among storage areas depending on whether or not the REORGANIZE argument is specified, see the *Oracle Rdb7 Guide to Database Design and Definition*.

**AREAS**
Specifies that the target of the data reorganization is storage areas. All rows are checked to see if they are in the correct storage area and if some are not, they are moved. This is the default.

**PAGES**
Specifies that the target of the data reorganization is database pages. All rows are checked if they are in the correct storage area and if some are not, they are moved. Then, all rows are checked if any should be moved within each storage area, and these rows are moved if there is space on or closer to the new target page.

**store-clause**
A new storage map definition that replaces the existing storage map. The store-clause allows you to specify which storage area files will be used to store rows from the table. Note that:

- All rows of a table can be associated with a single storage area.
- Rows of a table can be distributed among several storage areas.

- Rows of a table can be systematically distributed (horizontally partitioned) among several storage areas by specifying upper limits on the values for a column in a particular storage area.

The store-clause specifies only how you want to associate rows with areas and not the manner in which rows are assigned to pages within an area.

See the CREATE STORAGE MAP Statement for a description of the syntax for the store-clause. However, the effect of the clause in the ALTER STORAGE MAP statement depends on how you change the existing storage map.

**PARTITIONING IS UPDATABLE**
Specifies that the partitioning key can be modified. The partitioning key is the column or list of columns specified in the STORE USING clause.

See the *Oracle Rdb7 Guide to Database Design and Definition* for more information regarding partitioning.

**threshold-clause**
Specifies SPAM thresholds for logical areas with uniform format pages.

When you specify the THRESHOLD clause without enclosing it in parentheses, you are specifying the default threshold values for all areas specified in the ALTER STORAGE MAP statement. Because you cannot specify THRESHOLD values for existing storage areas, do not use this statement unless all areas specified in the ALTER STORAGE MAP statement are new areas.

To specify threshold values for a particular storage area, specify the clause as part of the STORE clause and enclose the THRESHOLD clause in parentheses. You can only specify threshold values for new areas, not existing ones.

For examples of specifying the THRESHOLD clause, see the *Oracle Rdb7 Guide to Database Design and Definition*. See the CREATE STORAGE MAP Statement for a description of the THRESHOLDS clause.

**STORE LISTS IN area-name**
Directs the database system to store the lists from tables in a specified storage area. You can store lists from different tables in the same area. You can create only one storage map for lists within each database.

You must specify RDB$SYSTEM as the default storage area for lists.

For more information, see the CREATE STORAGE MAP Statement.

**FOR (table-name)**
Specifies the table or tables to which this storage map applies. The named table must already be defined. If you want to store lists of more than one table in the storage area, separate the names of the tables with commas. For each area, you can specify one FOR clause and list of table names.

**FOR (table-name.col-name)**
Specifies the name of the table and column containing the list to which this storage map applies. Separate the table name and the column name with a period (.). The named table and column must already be defined. If you want to store multiple lists in the storage area, separate the table name and column name combinations with commas. For each area, you can specify one FOR clause and a list of column names.

**FILL RANDOMLY**
**FILL SEQUENTIALLY**
Specifies whether to fill the area set randomly or sequentially. Specifying FILL RANDOMLY or FILL SEQUENTIALLY requires a FOR clause. When a storage area is filled, it is removed from the list of available areas. Oracle Rdb does not attempt to store any more lists in that area during the current database attach. Instead, Oracle Rdb starts filling the next specified area.

When a set of areas is filled sequentially, Oracle Rdb stores lists in the first specified area until that area is filled. Use sequential filling when storing lists in write-once storage areas in a jukebox environment to avoid excess swapping of platters. In a jukebox environment, the filled storage area is marked with a FULL flag and the platter on which the area resides is no longer swapped in.

If the set of areas is filled randomly, lists are stored across multiple areas. This is the default. Random filling is intended for read/write media, which will benefit from the I/O distribution across the storage areas.

The keywords FILL RANDOMLY and FILL SEQUENTIALLY can only be applied to areas contained within an area list.

## Usage Notes

- Attempts to alter a storage map fail if that storage map refers to a table that is involved in a query at the same time. Users must detach from the database with a DISCONNECT statement before you can alter the storage map. When Oracle Rdb first accesses an object, such as the storage map, a lock is placed on that object and not released until the user exits the database. If you attempt to update this object, you get a LOCK CONFLICT ON CLIENT message due to the other user's access to the object.

Similarly, while you alter a storage map, users cannot execute queries involving tables that a storage map refers to until you completed the transaction with a COMMIT or ROLLBACK statement for the ALTER statement. The user receives a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system tables. (System tables contain information about objects in the database.) Therefore, attempts to update an object lock out attempts to query that object. These locks are held until the DDL operation is committed or rolled back.

The WAIT/NOWAIT clause of the SET TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify SET TRANSACTION WAIT for the metadata update transaction, you get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource;
no-wait parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
SQL>
```

However, a user's query waits for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT in the SET TRANSACTION statement.

- You must specify either a store-clause, a PLACEMENT VIA INDEX clause, a REORGANIZE clause, or a COMPRESSION clause in an ALTER STORAGE MAP statement. You can specify a PLACEMENT VIA INDEX clause, a REORGANIZE clause, or a COMPRESSION clause in any order. When the REORGANIZE clause is used, rows are moved and assigned to new database keys.
- You must execute the ALTER STORAGE MAP statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with characteristics specified in the most recent DECLARE TRANSACTION statement.
- The ALTER STORAGE MAP statement fails when both of the following circumstances are true:
  - The database to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The database was declared using the FILENAME argument.

Under these circumstances, the ALTER STORAGE MAP statement fails with the following error when you issue it:

```
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-CDDISREQ, CDD required for metadata updates
                  is not being maintained
```

- The following notes describe the behavior of the REORGANIZE clause:
  - If storage areas were named in the original storage map that are not named in the new storage map, rows in those storage areas deleted from the original storage map are moved to storage areas specified by the new storage map.
  - If the new storage map definition specifies the REORGANIZE AREAS clause, the database software checks all other rows to determine whether or not they are in the correct storage area. If the rows are not in the correct storage area, they are deleted from their current storage area and stored in the correct one.
  - If the ALTER STORAGE MAP statement specifies a REORGANIZE PAGES clause, the database software checks which rows can be moved to the pages where they would be placed if they were being stored as new rows. If the rows fit on those preferred pages or pages closer to the preferred pages than they currently are, they are moved.
  - If the new storage map definition includes the WITH LIMIT OF clause when you specify the REORGANIZE clause, all rows are read and stored again, whether or not you give new values.
  - If the new storage map definition includes only the COMPRESSION clause, all rows are read, the compression characteristics are changed, and all rows are stored again, whether or not you specify the REORGANIZE clause.
  - If the new storage map definition includes the PLACEMENT VIA INDEX clause when you specify the REORGANIZE clause, only the new rows based on the new index name are stored.
  - If the new storage map definition includes the USING column-name clause when you specify the REORGANIZE clause, only the new rows based on the new column name are stored.
  - The REORGANIZE clause works for one or more areas in the storage maps.

- If you do not specify the REORGANIZE clause as part of the ALTER STORAGE MAP statement and the new storage map definition omits the name of a storage area that was in the original storage map definition, Oracle Rdb treats the database rows in the following ways:
  - The rows are unloaded from the omitted storage area to the specified areas, according to the new storage map.
  - The rows are stored into the named storage areas according to the specified WITH LIMIT OF clause.
  - The rows are compressed according to the characteristics specified in the COMPRESSION clause.
- Do not use the ALTER STORAGE MAP statement to reorganize or otherwise modify read-only storage areas. If a storage area was designated as read-only, you must change it to a read/write storage area before using the ALTER STORAGE MAP statement to modify it.
- You can store lists and tables in separate storage areas.
- If a list storage map refers to storage area AREA1, you cannot delete AREA1. You can, however, add another storage area.
- If you repeat a column or table in the storage map with a different area, then all columns of data type LIST OF BYTE VARYING are stored randomly across the specified areas.
- If a storage map does not contain an overflow partition (defined by the OTHERWISE clause), you can alter the storage map and add new partitions without reorganizing the storage areas. For more information, see the Usage Notes in the CREATE STORAGE MAP Statement.
- If a storage map contains an overflow partition and you want to alter the storage map to rid it of the overflow partition, you do not need to use the REORGANIZE clause. Oracle Rdb moves the existing data to the appropriate storage area.
- If a storage map contains an overflow partition and you want to alter the storage map to change the overflow partition to a partition defined with the WITH LIMIT OF clause, you must use the REORGANIZE clause if you want existing data that is stored in the overflow partition moved to the appropriate storage area.

  For more information about omitting overflow partitions (and altering storage maps in general), see the *Oracle Rdb7 Guide to Database Design and Definition*.

- Oracle Rdb checks to ensure that list maps are not created on system tables. This check can only be done on data definition statements executed after an ATTACH statement. This check cannot be done when an attach is performed by the CREATE DATABASE or IMPORT statements because the map is created before the referenced list objects exist.
- You can only modify a storage map from PARTITIONING IS NOT UPDATABLE to PARTITIONING IS UPDATABLE. You cannot do the reverse because the data may no longer be strictly partitioned according to the criteria specified in the STORE USING clause.

## Examples

Example 1: Reorganizing storage area data using the ALTER STORAGE MAP statement

The following example defines a new storage area, EMPIDS_MID2, to handle the employee ID numbers from 601 to 900 and to reorganize the data from an existing storage area, EMPIDS_OVER. The current data that is stored in employee ID numbers from 601 to 900 is moved according to the new limits. Because no AREA or PAGE option is specified, the default method of reorganization is by storage areas.

```
SQL> ALTER DATABASE FILENAME mf_personnel ADD STORAGE AREA
cont> EMPIDS_MID2 PAGE FORMAT IS MIXED;
SQL> ATTACH 'FILENAME mf_personneL';
SQL> ALTER STORAGE MAP EMPLOYEES_MAP
cont>  STORE USING (EMPLOYEE_ID)
cont>       IN EMPIDS_LOW WITH LIMIT OF ('00300')
cont>       IN EMPIDS_MID WITH LIMIT OF  ('00600')
cont>       IN EMPIDS_MID2 WITH LIMIT OF ('00900')
cont>          OTHERWISE IN EMPIDS_OVER
cont>          REORGANIZE;
```

Example 2: Changing the logical area thresholds with an ALTER STORAGE MAP statement

The following example defines a new storage map, UNIFORM1_MAP, and specifies thresholds for the logical area in the UNIFORM1 storage area. The ALTER STORAGE MAP statement is used to enable row compression.

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>  ADD STORAGE AREA UNIFORM1;
SQL> ATTACH 'FILENAME mf_personnel';
SQL> CREATE TABLE TEST (COL1 REAL);
SQL> CREATE STORAGE MAP UNIFORM1_MAP FOR TEST
cont>  STORE IN UNIFORM1
cont>    (THRESHOLDS ARE (80,90,95));
SQL> ALTER STORAGE MAP UNIFORM1_MAP
cont>  STORE IN UNIFORM1
cont>  ENABLE COMPRESSION;
```

Example 3: Changing an overflow partition to a WITH LIMIT OF partition

To change the overflow partition to a partition defined with the WITH LIMIT OF clause, you must use the REORGANIZE clause if you want existing data that is stored in the overflow partition moved to the appropriate storage area. For example, suppose the JOB_HISTORY table contains a row with an EMPLOYEE_ID of 10001 and the JH_MAP storage map is defined, as shown in the following example:

```
SQL> SHOW STORAGE MAP JH_MAP
     JH_MAP
 For Table:            JOB_HISTORY
 Compression is:       ENABLED
 Store clause:         STORE USING (EMPLOYEE_ID)
                             IN PERSONNEL_1 WITH LIMIT OF ('00399')
                             IN PERSONNEL_2 WITH LIMIT OF ('00699')
                       OTHERWISE IN PERSONNEL_3
SQL>
```

If you want to change the PERSONNEL_3 storage area from an overflow partition to a partition with a limit of 10,000 and add the partition PERSONNEL_4, you must use the REORGANIZE clause to ensure that Oracle Rdb moves existing rows to the new storage area. The following example shows the ALTER STORAGE MAP statement that accomplishes this change:

```
SQL> ALTER STORAGE MAP JH_MAP
cont>      STORE USING (EMPLOYEE_ID)
cont>            IN PERSONNEL_1 WITH LIMIT OF ('00399')
cont>            IN PERSONNEL_2 WITH LIMIT OF ('00699')
cont>            IN PERSONNEL_3 WITH LIMIT OF ('10000')
cont>            IN PERSONNEL_4 WITH LIMIT OF ('10399')
cont>      REORGANIZE;
SQL>
```

## ALTER TABLE Statement

Changes an existing table definition. You can:

- Add columns
- Add constraints to tables or columns
- Modify columns
- Modify character sets
- Modify data types
- Delete columns
- Delete constraints

The ALTER TABLE statement can also add or delete table-specific constraints, updating the physical database appropriately. These constraints can be deleted, declared, or both. You cannot alter an existing constraint; instead, you must specifically delete it by name and then create it again with the definition you desire. You can display the names for all constraints currently associated with a table by using the SHOW TABLE statement. Any number of constraints can be deleted and declared at both the table and column levels.

When you execute this statement, SQL modifies the named column definitions in the table definition. All the columns that you do not mention remain the same. SQL defines new versions of columns *before* defining constraints. Then, SQL defines and evaluates constraints before storing them. Therefore, if columns and constraints are defined in the same table definition, constraints always apply to the latest version of a column.

When you change a table definition, other users see the revised definition only when they declare the schema after you commit the changes.

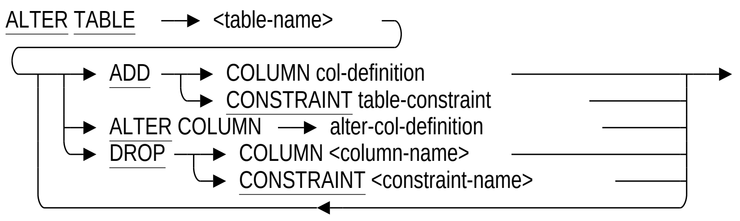
### Environment

You can use the ALTER TABLE statement:

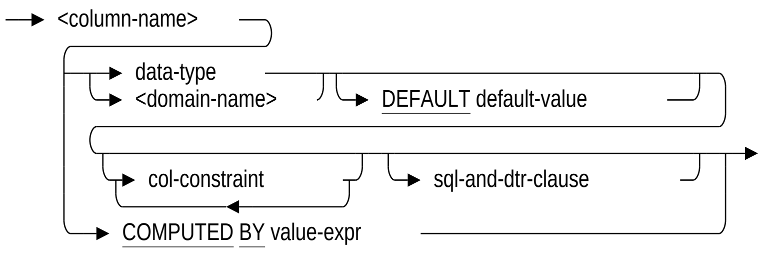
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

## ALTER TABLE Statement

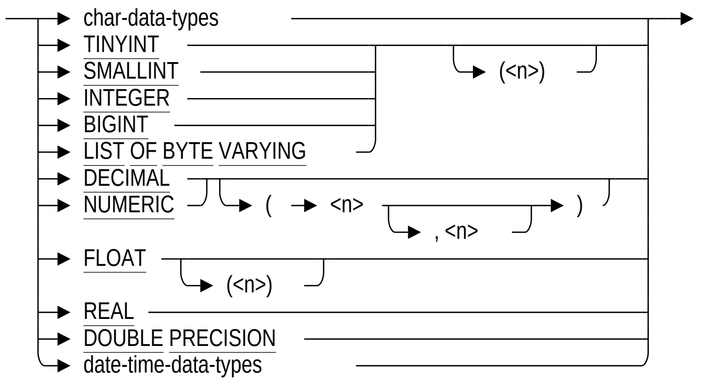
### Format



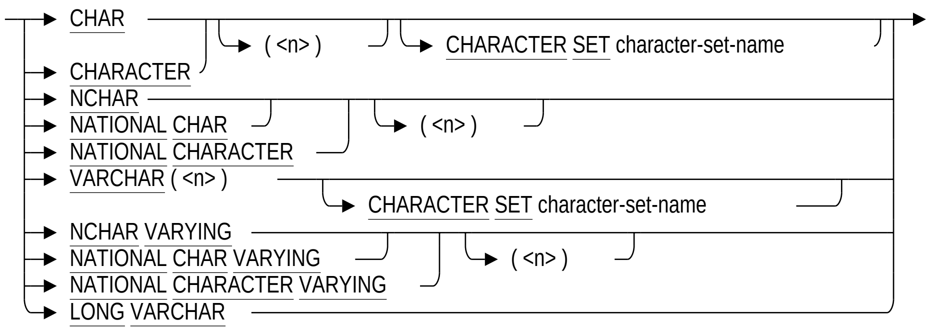
col-definition =



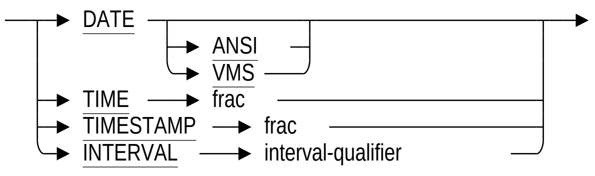
data-type =



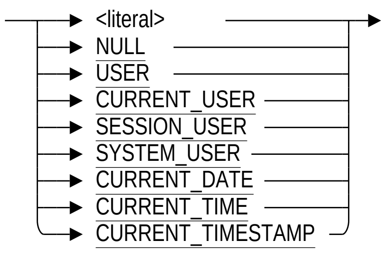
char-data-types =



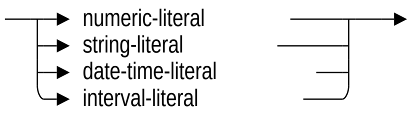
date-time-data-types =



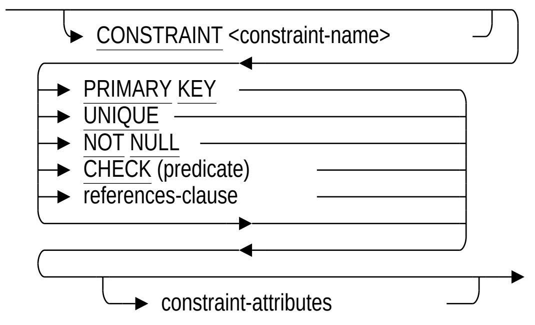
default-value =



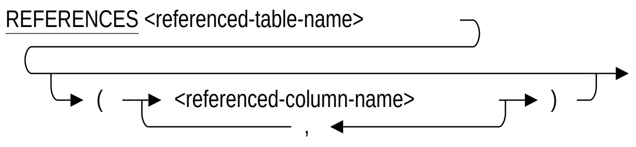
literal =



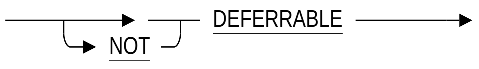
col-constraint=



references-clause =



constraint-attributes =



sql-and-dtr-clause =

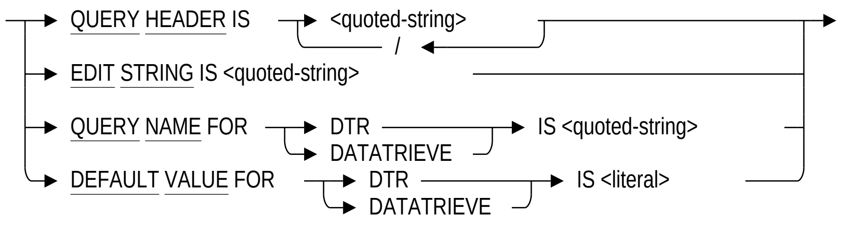


table-constraint =

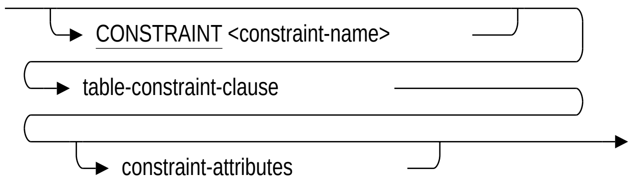
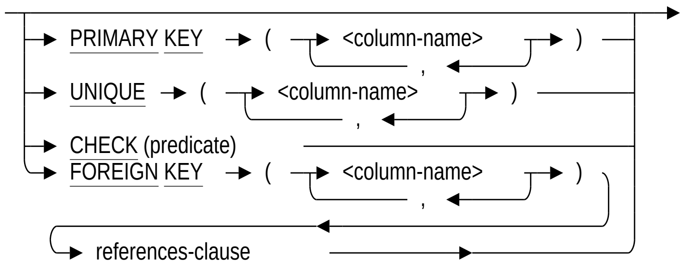
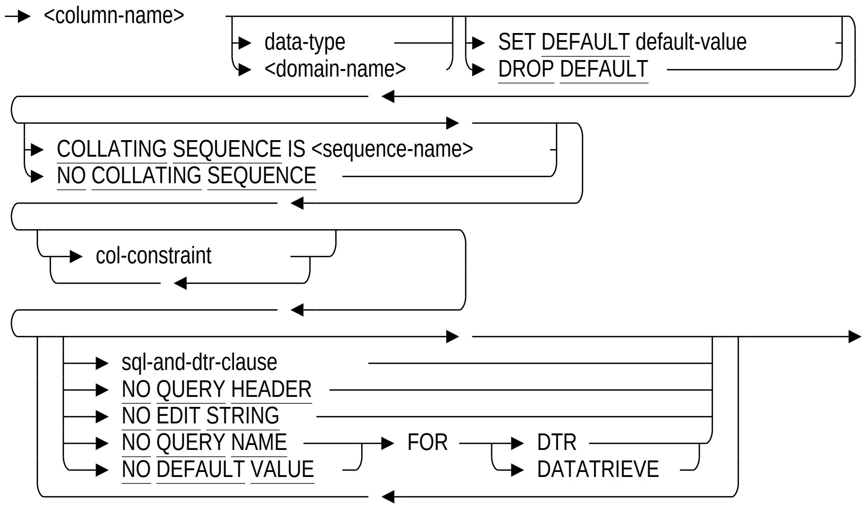


table-constraint-clause =



alter-col-definition =



## Arguments

**table-name**
The name of the table whose definition you want to change.

**ADD COLUMN col-definition**
Creates an additional column in the table. SQL adds the column to the right of the existing columns in the table. The column definition specifies a data type or domain name, optional default value, optional column constraints, and optional formatting and DATATRIEVE clauses.

The COLUMN keyword is optional.

**column-name**
The name of a column you want to create in the table. You need to specify a column name whether you directly specify a data type in the column definition, or indirectly specify a data type by naming a domain in the column definition.

**data-type**
A valid SQL data type. Specifying an explicit data type to associate with a column is an alternative to specifying a domain name.

See Section 2.3 for more information on data types.

Using the ALTER clause to change the data type of a column (directly or indirectly by specifying a domain) requires caution:

- If you change a column to a data type with a larger capacity, or increase the scale factor for a column, or change the character set, you may have to modify source programs that refer to the column and precompile them again.
- If you change a column to a data type with a smaller capacity, SQL truncates values already stored in the database that exceed the capacity of the new data type, but only when it retrieves those values. (The values are not truncated in the database, however, until they are updated. If you only retrieve data, you can change the data type back to the original, and SQL again retrieves the entire original value.)
- You can change a DATE column only to a character data type (CHAR, VARCHAR, LONG VARCHAR, NCHAR, NATIONAL CHAR, NCHAR VARYING, or NATIONAL CHAR VARYING). If you attempt to change a DATE column to anything but a character data type, SQL returns an error message.

**char-data-types**
A valid SQL character data type. See Section 2.3.1 for more information on character data types.

**date-time-data-types**
A valid SQL date-time data type. See Section 2.3.5 for more information on date-time data types.

**domain-name**
The name of a domain created in a CREATE DOMAIN statement. SQL gives the column the data type specified in the domain. For more information on domains, see the CREATE DOMAIN Statement.

For most purposes, specify a domain instead of an explicit data type.

- Domains ensure that columns in multiple tables that serve the same purpose all have the same data type. For example, several tables in the sample personnel database refer to the domain ID_DOM.
- A domain lets you change the data type for all the columns that refer to it in one operation by changing the domain itself with an ALTER DOMAIN statement. For example, if you want to change the data type for the column EMPLOYEE_ID from CHAR(5) to CHAR(6), you need only alter the data type for ID_DOM. You do not have to alter the data type for the column EMPLOYEE_ID in the tables DEGREES, EMPLOYEES, JOB_HISTORY, or SALARY_HISTORY, nor do you have to alter the column MANAGER_ID in the DEPARTMENTS table.

However, you might not want to use domains when you create tables if:

- Your application must be compatible with the current ANSI/ISO SQL standard. Domains are not part of the ANSI/ISO 1989 standard; however, domains are part of the ANSI/ISO SQL standard.
- You are creating tables that do not need the advantages of domains.

**DEFAULT default-value**
Provides a default value for a column if the row that is inserted does not include a value for that column. You can use literals, the NULL keyword, the user name, the session user name, the system user name, the current date, the current time, or the current timestamp as default values. For more information about NULL, see Section 2.6.1 and the Usage Notes following this Arguments list.

You can add a default value to an existing column or alter the existing default value of a column by altering the table. However, doing so has no effect on the values stored in existing rows.

If you do not specify a default value, a column inherits the default value from the domain. If you do not specify a default value for either the column or domain, SQL assigns NULL as the default value.

If you specify a default value for either the column or domain when a column is added, SQL propagates the default value from the column or domain to all previously stored rows. Therefore, when you add a column to a table and specify a default value for the column, SQL stores the default value in the newly added column of all the previously stored rows. Likewise, if the newly added column is based upon a domain that specifies a default value, SQL stores the default value in the column of all the previously stored rows.

Example 6–1 shows that SQL stores the default value in the column when you add a column that specifies a default value.

**Example 6–1 Adding Columns with Default Values to Tables**

```
SQL> -- Add the column PHONE and specify a default value.
SQL> --
SQL> ALTER TABLE EMPLOYEES
cont>     ADD PHONE CHAR(7) DEFAULT 'None';
SQL> --
SQL> -- The result table shows that the rows contain the default value
SQL> -- of the PHONE column.
SQL> --
SQL> SELECT LAST_NAME, PHONE FROM EMPLOYEES;
 LAST_NAME        PHONE
 Toliver          None
 Smith            None
 Dietrich         None
 Kilpatrick       None
    .
    .
    .
SQL>
```

Because SQL updates data when you add a column with a default value other than NULL, the ALTER TABLE statement can take some time to complete when the table contains many rows. (If you specify a default value of NULL, SQL does not modify the data because SQL automatically returns a null value for columns that have no actual value stored in them.) If you want to add more than one column with default values, add them in one ALTER TABLE statement. When you do so, SQL scans the table data once instead of many times.

Because data is added to the rows, adding a column with a default value may result in fragmented records. For information about locating and correcting record fragmentation, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

Remember that the default value for a column is not the same as the missing value that you can specify using the RDO interface. See the *Oracle Rdb7 Guide to Database Design and Definition* for information on the difference between a default value and a missing value.

**default-value**
Specifies the default value of a column. The following table lists the valid values:

| Default Value | Description |
|---|---|
| literal | A value expression. Literal values can be numeric, character string, or date data types. |
| NULL | A null value. |
| USER | The current, active user name for a request. |
| CURRENT_USER | The current, active user name for a request. If a definer's rights request is executing, SQL returns the definer's user name. If not, SQL returns the session user name, if it exists. Otherwise, SQL returns the system user name. |
| SESSION_USER | The current, active session user name. If the session user name does not exist, SQL returns the system user name. |
| SYSTEM_USER | The user name of the process at the time of the database attach. |
| CURRENT_DATE | The DATE data type value containing year, month, and day for date "today". |
| CURRENT_TIME | The TIME data type value containing hours, minutes, and seconds for time "now". |
| CURRENT_ TIMESTAMP | The date and time currently defined in Oracle Rdb. |

**literal**
Specifies a literal value. For more information, see Section 2.4.

#### col-constraint

Specifies a constraint that column values inserted into the table must satisfy. You can specify more than one column constraint. For example:

```
SQL> ALTER TABLE EMPLOYEE
cont>   ADD ID_NUMBER INT NOT NULL UNIQUE;
```

You can name each constraint. For example:

```
SQL> ALTER TABLE EMPLOYEE
cont>   ADD ID_NUMBER INT
cont>   CONSTRAINT A NOT NULL
cont>   CONSTRAINT B UNIQUE;
```

#### CONSTRAINT constraint-name

Names the column constraint.

#### PRIMARY KEY

Declares this column to be a primary key. SQL requires that values in a primary key column be unique and not null; therefore, you do not need to specify the UNIQUE and NOT NULL column constraints for a primary key column.

#### UNIQUE

Specifies that values in the associated column must be unique.

#### NOT NULL

Restricts values in the column to values that are not null.

#### CHECK (predicate)

Specifies a predicate that column values inserted into the table must satisfy. See Section 2.7 for details on specifying predicates.

Predicates in CHECK column constraints can only refer directly to the column with which they are associated. See the Usage Notes for the CREATE TABLE Statement for details.

#### references-clause

Specifies the name of a column or columns that are a unique key or a primary key in the referenced table. When the REFERENCES clause is specified as a column constraint, the column name specified in the col-definition clause becomes a foreign key for the table being defined (the referencing table). When the REFERENCES clause is selected as a table constraint, the column name or column names specified in the FOREIGN KEY clause become a foreign key for the referencing table.

**REFERENCES referenced-table-name**
Specifies the name of the table that contains the unique key or primary key referred to by the referencing table. You must have the SQL REFERENCES or CREATE privileges on the referenced table to declare a constraint that refers to another table.

**referenced-column-name**
For a column constraint, the name of the column that is a unique key or a primary key in the referenced table. For a table constraint, the referenced column name is the name of the column or columns that are a unique key or primary key in the referenced table. If you omit the referenced-column-name clause, the primary key is selected by default.

**constraint-attributes**
There are two constraint attributes: DEFERRABLE and NOT DEFERRABLE.

Specifying NOT DEFERRABLE means that evaluation of the constraint must take place when the INSERT, DELETE, or UPDATE statement executes.

Specifying DEFERRABLE means that evaluation of the constraint can take place at any later time. Unless otherwise specified, evaluation of the constraint takes place as the COMMIT statement executes. You can use the SET ALL CONSTRAINTS statement to have all constraints evaluated earlier. See the SET ALL CONSTRAINTS Statement for more information.

If you are using the default SQLV40 dialect, the default constraint attribute is DEFERRABLE. When using this dialect, Oracle Rdb displays a deprecated feature message for all constraints defined without specification of one of the constraint attributes. If you are using the SQL92 dialect, the default is NOT DEFERRABLE.

**sql-and-dtr-clause**
Optional SQL and DATATRIEVE formatting clause. See Section 2.5 for more information.

If you specify a formatting clause for a column that is based on a domain that also specifies a formatting clause, the formatting clause in the table definition overrides the one in the domain definition.

**COMPUTED BY value-expr**
Specifies that the value of this column is calculated from values in other columns and constant expressions. See the CREATE TABLE Statement for more information.

### ADD CONSTRAINT table-constraint

Adds a table constraint definition. The four types of table constraints are PRIMARY KEY, UNIQUE, CHECK, and FOREIGN KEY.

### CONSTRAINT constraint-name

The CONSTRAINT clause specifies a name for the table constraint. The name is used for a variety of purposes:

- The INTEG_FAIL error message specifies the name when an INSERT, UPDATE, or DELETE statement violates the constraint.
- The ALTER TABLE DROP CONSTRAINT statements specify the constraint name.
- The SHOW TABLE statements display the names of constraints.
- The EVALUATING clause of the SET and the DECLARE TRANSACTION statements specifies constraint names.

The CONSTRAINT clause is optional. If you omit the constraint name, SQL creates a name. However, Oracle Rdb recommends that you always name column and table constraints. The constraint names generated by SQL may be obscure and, in programs, may change between compile time and run time. If you supply a constraint name with the CONSTRAINT clause, the name must be unique in the schema.

### PRIMARY KEY column-name

Used to declare columns as a primary key for the table being altered. Any foreign key that refers to this column must refer to this primary key.

### UNIQUE column-name

The name of columns in the table being defined that are part of a unique key.

### CHECK (predicate)

A predicate that column values inserted into the table must satisfy.

Predicates in CHECK column constraints can refer directly only to the column with which they are associated. See Section 2.7 for details on specifying predicates.

Predicates in CHECK table constraints can refer to any column in the table. Column select expressions within the predicate can refer to other tables in the schema.

See the CREATE TABLE Statement for additional details on CHECK constraints.

### FOREIGN KEY column-name

The name of a column or columns that you want to declare as a foreign key in the table you are altering (the referencing table).

### references-clause

Specifies the name of the column or columns that are a unique key or primary key in the referenced table. When the REFERENCES clause is selected as a table constraint, the column names specified in the FOREIGN KEY clause become a foreign key for the referencing table.

### constraint-attributes

There are two constraint attributes: DEFERRABLE and NOT DEFERRABLE.

For more information, see the constraint-attributes argument described earlier in this Arguments list.

### ALTER COLUMN alter-col-definition

Modifies the column specified by the column name. The COLUMN keyword is optional.

You can modify some elements of a column definition but not others. You cannot change an existing column constraint. However, you can delete the existing constraint and add a new column constraint using the alter-col-definition clause to achieve the same result.

### column-name

The name of the column being modified.

### data-type

An explanation of the data type argument appears earlier in this Arguments list.

Using the ALTER clause to change the data type of a column (directly or indirectly by specifying a domain) requires caution:

- If you change a column to a data type with a larger capacity or increase the scale factor for a column, you may have to modify source programs that refer to the column and precompile them again.
- If you change a column to a data type with a smaller capacity, SQL truncates values already stored in the database that exceed the capacity of the new data type, but only when it retrieves those values. (The values are not truncated in the database, however, until they are updated. If you only retrieve data, you can change the data type back to the original, and SQL again retrieves the entire original value.)

- You can change a DATE column only to a character data type (CHAR, VARCHAR, or LONG VARCHAR). If you change a DATE column to anything but a character data type, you could get unexpected results.

**domain-name**
An explanation of the domain-name argument appears earlier in this Arguments list.

**SET DEFAULT default-value**
Specifies a default value for the column.

An explanation of the default-value argument appears earlier in this Arguments list.

**DROP DEFAULT**
Deletes (drops) the default value of a column in a table.

**col-constraint**
Specifies the constraint you are defining for an existing column. The syntax and explanation are described earlier in this Arguments list.

**sql-and-dtr-clause**
Optional SQL and DATATRIEVE formatting clause. See Section 2.5 for more information.

If you specify a formatting clause for a column that is based on a domain that also specifies a formatting clause, the formatting clause in the table definition overrides the one in the domain definition.

**DROP COLUMN column-name**
Deletes the specified column. The COLUMN keyword is optional.

**DROP CONSTRAINT constraint-name**
Deletes the specified column constraint or table constraint from the table definition.

## Usage Notes

- Attempts to alter a table fail if that table is involved in a query at the same time. Users must detach from the database with a DISCONNECT statement before you can alter the table. When Oracle Rdb first accesses an object, such as the table, a lock is placed on that object and not released until the user exits the database. If you attempt to update this object, you will get a LOCK CONFLICT ON CLIENT message due to the other user's access to the object.

Similarly, while you alter a table, users cannot execute queries involving that table until you completed the transaction with a COMMIT or ROLLBACK statement for the ALTER TABLE statement. The user receives a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system tables. (System tables contain information about objects in the database.) Therefore, attempts to update an object lock out attempts to query that object. These locks are held until the DDL operation is committed or rolled back.

The WAIT/NOWAIT clause of the SET TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify SET TRANSACTION WAIT for the metadata update transaction, you get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource;
no-wait parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
SQL>
```

However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT in the SET TRANSACTION statement.

- You must execute the ALTER TABLE statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with characteristics specified in the most recent DECLARE TRANSACTION statement.
- You can only alter table definitions. You cannot alter view definitions.
- Because Oracle Rdb creates dependencies between stored procedures and metadata (like tables) on which they are compiled and stored, adding a column with a language semantic dependency causes the stored procedure in which the column resides to be invalidated. See the CREATE MODULE Statement for a list of ALTER TABLE statements that can or cannot cause stored procedure invalidation.

  See the *Oracle Rdb7 Guide to SQL Programming* for detailed information about stored procedure dependency types and how metadata changes can cause invalidation of stored procedures.
- You cannot delete or alter a column in a table if:
  - That column is referred to by a view.

  – An index is based on that column.
  – The schema contains a constraint definition (other than NOT NULL) that refers to the column.

  You can delete or alter a column if you first delete the view, index, or constraint that refers to the column.

- You can alter the data type of a column with a referencing NOT NULL constraint without first deleting the constraint.
- You can use the ALTER TABLE statement to add or modify the default value for a column.

  You can use a default value such as NULL or "Not Applicable" that clearly demonstrates that no data was inserted into a column. If a column would usually contain a particular value, you can use that value as the default. For example, if most company employees work full-time, you could make full-time the default value for a work status column.

  If you specify a default value for a column that you base on a domain and you specified a default value for that domain, the default value for the column overrides the default value for the domain.

  To remove a default value, use the DROP DEFAULT clause, as follows:

  ```
  SQL> ALTER TABLE EMPLOYEES
  cont>   ALTER BIRTHDAY
  cont>   DROP DEFAULT;
  ```

  If you change or add a default value for a domain, the change has no effect on any existing data in the database; that is, the rows already stored in the database with columns that contain the *old* default value are not changed.

  Remember that the default value is *not* the same as the missing value that you can specify using the RDO interface. In contrast to default values, changing the missing value does change what is displayed by applications based on RDO for columns that have no data value stored and that have a missing value defined. See the *Oracle Rdb7 Guide to Database Design and Definition* for a description of the difference between a default value and a missing value.

- The result data type for USER, CURRENT_USER, SESSION_USER, and SYSTEM_USER keywords is CHAR(31).
- You can use the ALTER TABLE statement to add or delete column and table constraints.

  See the Usage Notes section in the CREATE TABLE Statement for details on the differences between column constraints and table constraints.

- The ALTER TABLE statement fails if you add a constraint and the condition is not true.
- You must delete and create the view definition again for views to display new columns. Existing view definitions do not display columns added with the ALTER TABLE statement. Views display only the columns that existed when the views were created.
- Changes you make to tables created with the FROM clause (based on a repository definition) or to tables based on domains created with the FROM clause can affect other schemas and applications. If the schema was declared with the PATHNAME clause, changes made with the ALTER TABLE . . . ADD or the ALTER TABLE . . . ALTER statement are immediately written to the repository record or field definitions. If the schema was declared with the FILENAME clause, the changes are written to the repository when the next INTEGRATE SCHEMA . . . ALTER DICTIONARY statement is issued.

  The changes affect applications and other schemas that use the same repository definition when the application recompiles or the database integrates with the repository.

  For this reason, use caution when altering tables that are based on repository definitions. Make sure that changes you make through ALTER TABLE statements will not have unintended effects on other users or applications that share the repository definitions.
- The ALTER TABLE statement fails when both of the following circumstances are true:
  - The schema to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The schema was declared using the FILENAME argument.

  Under these circumstances, the statement fails with the following error when you issue it:

  ```
  %RDB-E-NO_META_UPDATE, metadata update failed
  -RDMS-F-CDDISREQ, CDD required for metadata updates is not being maintained
  ```
- Constraints are evaluated at definition time when there is data in the table. You will not be able to add a constraint when rows exist that violate the constraint. If this check fails, you get an error message.
- You cannot execute the ALTER TABLE statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on the RDB$SYSTEM storage area.

- The ALTER TABLE statement allows you to change the character set associated with a column name. However, if this is done after data is entered into a table, SQL returns a data conversion error when you try to select rows from that table.
- You can specify the national character data type by using the NCHAR, NATIONAL CHAR, NCHAR VARYING, or NATIONAL CHAR VARYING data types. The national character data type is defined by the database national character set when the database is created. See Section 2.3 for more information regarding national character data types.
- You can specify the length of the data type in characters or octets. By default, data types are specified in octets. By preceding the ALTER TABLE statement with the SET CHARACTER LENGTH 'CHARACTERS' or SET DIALECT 'MIA' statement, you change the length to characters. See the SET CHARACTER LENGTH Statement for more information regarding the SET CHARACTER LENGTH and SET DIALECT statements.
- A computed by column is set to NULL if it references a table that has been deleted by a DROP TABLE table-name CASCADE statement. For example:

```
SQL> CREATE TABLE t1 (col1 INTEGER,
cont>                 col2 INTEGER);
SQL> --
SQL> CREATE TABLE t2 (x INTEGER,
cont>                 y COMPUTED BY (SELECT COUNT(*) FROM
cont>                   t1 WHERE t1.col1 = t2.x));
SQL> --
SQL> -- Assume values have been inserted into the tables.
SQL> --
SQL> SELECT * FROM t1;
        COL1          COL2
           1           100
           1           101
           1           102
           2           200
           3           300
5 rows selected
SQL> SELECT * FROM t2;
           X             Y
           1             3
           3             1
2 rows selected
SQL> --
SQL> DROP TABLE t1 CASCADE;
SQL> SELECT * FROM t2;
           X             Y
           1             NULL
           3             NULL
```

You can either alter the computed by column to have a new data type or value, or delete that column from the table.

## Examples

Example 1: Adding a column to the EMPLOYEES table

```
SQL> ALTER TABLE EMPLOYEES ADD SALARY INTEGER;
```

Example 2: Adding a column and altering a column in the COLLEGES table

The following example adds two columns, one with a DATATRIEVE clause that specifies a query name to the COLLEGES table. It also modifies the data type of the POSTAL_CODE column to accept 9 characters instead of 5 characters:

```
SQL> SHOW TABLE COLLEGES;
Information for table COLLEGES

Comment on table COLLEGES:
names and addresses of colleges attended by employees

Columns for table COLLEGES:
Column Name                     Data Type        Domain
-----------                     ---------        ------
COLLEGE_CODE                    CHAR(4)          COLLEGE_CODE_DOM
 Primary Key constraint COLLEGES_PRIMARY_COLLEGE_CODE
COLLEGE_NAME                    CHAR(25)         COLLEGE_NAME_DOM
CITY                            CHAR(20)         CITY_DOM
STATE                           CHAR(2)          STATE_DOM
POSTAL_CODE                     CHAR(5)          POSTAL_CODE_DOM
   .
   .
   .
SQL> ALTER TABLE COLLEGES
cont>   ADD RANKING INTEGER
cont>   ADD NUMBER_ALUMS INTEGER
cont>           QUERY_NAME FOR DTR IS 'ALUMS';
SQL>   ALTER DOMAIN POSTAL_CODE_DOM CHAR(9);
SQL> SHOW TABLE COLLEGES;

Information for table COLLEGES

Comment on table COLLEGES:
names and addresses of colleges attended by employees
```

```
Columns for table COLLEGES:
Column Name                     Data Type        Domain
-----------                     ---------        ------
COLLEGE_CODE                    CHAR(4)          COLLEGE_CODE_DOM
 Primary Key constraint COLLEGES_PRIMARY_COLLEGE_CODE
COLLEGE_NAME                    CHAR(25)         COLLEGE_NAME_DOM
CITY                            CHAR(20)         CITY_DOM
STATE                           CHAR(2)          STATE_DOM
POSTAL_CODE                     CHAR(9)          POSTAL_CODE_DOM
RANKING                         INTEGER
NUMBER_ALUMS                    INTEGER
 Query Name:    ALUMS
   .
   .
   .
```

Example 3: Adding and modifying default values

```
SQL> -- Add a default value to the column HOURS_OVERTIME.
SQL> --
SQL> CREATE TABLE DAILY_SALES (HOURS_OVERTIME INT,
cont> HOURS_WORKED INT, GROSS_SALES INT, SALESPERSON CHAR (10));
SQL> --
SQL> -- Change the default value for the column HOURS_OVERTIME.
SQL> --
SQL> ALTER TABLE DAILY_SALES
cont> ALTER HOURS_OVERTIME
cont> SET DEFAULT 0;
SQL> --
SQL> -- Insert the day’s sales figures into the table,
SQL> -- accepting the default values for the SALESPERSON,
SQL> -- HOURS_WORKED, and HOURS_OVERTIME columns.
SQL> --
SQL> INSERT INTO DAILY_SALES
cont> (GROSS_SALES)
cont>  VALUES
cont>  (2567);
SQL> INSERT INTO DAILY_SALES
cont> (SALESPERSON)
cont> VALUES
cont> (’BARTLETT’);
SQL> SELECT * FROM DAILY_SALES;

 SALESPERSON      HOURS_WORKED   HOURS_OVERTIME   GROSS_SALES
 BARTLETT                    9                0          2567
1 row selected
```

Example 4: Deleting a constraint from the EMPLOYEES table

```
SQL> -- To find out the name of a constraint, use the
SQL> -- SHOW TABLES statement. The SHOW TABLES
SQL> -- statement shows all constraints that refer to a table,
SQL> -- not just those defined as part of the table’s
SQL> -- definition. For that reason it is good practice to
SQL> -- always use a prefix to identify the table
SQL> -- associated with a constraint when you assign
SQL> -- constraint names with the CONSTRAINT clause.
SQL> --
SQL> -- The constraint DEGREES_FOREIGN1 in this SHOW
SQL> -- display follows that convention to indicate that
SQL> -- the constraint is associated with the DEGREES, not
SQL> -- the EMPLOYEES, table despite the constraint’s
SQL> -- presence in the EMPLOYEES display.
SQL> SHOW TABLE EMPLOYEES
Information for table EMPLOYEES

Comment on table EMPLOYEES:
personal information about each employee

Columns for table EMPLOYEES:
Column Name                     Data Type        Domain
-----------                     ---------        ------
EMPLOYEE_ID                     CHAR(5)          ID_DOM
 Primary Key constraint EMPLOYEES_PRIMARY_EMPLOYEE_ID
LAST_NAME                       CHAR(14)         LAST_NAME_DOM
FIRST_NAME                      CHAR(10)         FIRST_NAME_DOM
MIDDLE_INITIAL                  CHAR(1)          MIDDLE_INITIAL_DOM
ADDRESS_DATA_1                  CHAR(25)         ADDRESS_DATA_1_DOM
ADDRESS_DATA_2                  CHAR(20)         ADDRESS_DATA_2_DOM
CITY                            CHAR(20)         CITY_DOM
STATE                           CHAR(2)          STATE_DOM
POSTAL_CODE                     CHAR(5)          POSTAL_CODE_DOM
SEX                             CHAR(1)          SEX_DOM
BIRTHDAY                        DATE             DATE_DOM
STATUS_CODE                     CHAR(1)          STATUS_CODE_DOM

Table constraints for EMPLOYEES:
EMPLOYEES_PRIMARY_EMPLOYEE_ID
 Primary Key constraint
 Column constraint for EMPLOYEES.EMPLOYEE_ID
 Evaluated on COMMIT
 Source:
        EMPLOYEES.EMPLOYEE_ID PRIMARY KEY
```

```
EMP_SEX_VALUES
 Check constraint
 Table constraint for EMPLOYEES
 Evaluated on COMMIT
 Source:
        CHECK          (
                         SEX IN ('M', 'F', '?')
                         )
EMP_STATUS_CODE_VALUES
 Check constraint
 Table constraint for EMPLOYEES
 Evaluated on COMMIT
 Source:
        CHECK          (
                         STATUS_CODE IN ('0', '1', '2', 'N')
                         )
Constraints referencing table EMPLOYEES:
DEGREES_FOREIGN1
 Foreign Key constraint
 Column constraint for DEGREES.EMPLOYEE_ID
 Evaluated on COMMIT
 Source:
        DEGREES.EMPLOYEE_ID REFERENCES EMPLOYEES (EMPLOYEE_ID)
JOB_HISTORY_FOREIGN1
 Foreign Key constraint
 Column constraint for JOB_HISTORY.EMPLOYEE_ID
 Evaluated on COMMIT
 Source:
        JOB_HISTORY.EMPLOYEE_ID REFERENCES EMPLOYEES (EMPLOYEE_ID)
RESUMES_FOREIGN1
 Foreign Key constraint
 Column constraint for RESUMES.EMPLOYEE_ID
 Evaluated on COMMIT
 Source:
        RESUMES.EMPLOYEE_ID REFERENCES EMPLOYEES (EMPLOYEE_ID)

SALARY_HISTORY_FOREIGN1
 Foreign Key constraint
 Column constraint for SALARY_HISTORY.EMPLOYEE_ID
 Evaluated on COMMIT
 Source:
        SALARY_HISTORY.EMPLOYEE_ID REFERENCES EMPLOYEES (EMPLOYEE_ID)
   .
   .
   .
SQL> ALTER TABLE EMPLOYEES DROP CONSTRAINT EMP_SEX_VALUES;
```

Example 5: Adding a NOT NULL constraint to the EMPLOYEES table

```
SQL> ALTER TABLE EMPLOYEES
cont>   ALTER BIRTHDAY
cont>   CONSTRAINT E_BIRTHDAY_NOT_NULL
cont>   NOT NULL
cont>   DEFERRABLE;
```

If any rows in the EMPLOYEES table have a null BIRTHDAY column, the ALTER statement fails and none of the changes described in it will be made.

Example 6: Altering the character set of a table column

Assume the database was created specifying the database default character set and identifier character set as DEC_KANJI and the national character set as KANJI. Also assume the ROMAJI column was created in the table COLOURS specifying the identifier character set.

```
SQL> SET CHARACTER LENGTH 'CHARACTERS';
SQL> SHOW TABLE (COLUMNS) COLOURS;
Information for table COLOURS

Columns for table COLOURS:
Column Name                     Data Type        Domain
-----------                     ---------        ------
ENGLISH                         CHAR(8)          MCS_DOM
         DEC_MCS 8 Characters,  8 Octets
FRENCH                          CHAR(8)          MCS_DOM
         DEC_MCS 8 Characters,  8 Octets
JAPANESE                        CHAR(4)          KANJI_DOM
         KANJI 4 Characters,  8 Octets
ROMAJI                          CHAR(8)          DEC_KANJI_DOM
KATAKANA                        CHAR(8)          KATAKANA_DOM
         KATAKANA 8 Characters,  8 Octets
HINDI                           CHAR(8)          HINDI_DOM
         DEVANAGARI 8 Characters,  8 Octets
GREEK                           CHAR(8)          GREEK_DOM
         ISOLATINGREEK 8 Characters,  8 Octets
ARABIC                          CHAR(8)          ARABIC_DOM
         ISOLATINARABIC 8 Characters,  8 Octets
RUSSIAN                         CHAR(8)          RUSSIAN_DOM
         ISOLATINCYRILLIC 8 Characters,  8 Octets

SQL> ALTER TABLE COLOURS ALTER ROMAJI NCHAR(8);
SQL> SHOW TABLE (COLUMNS) COLOURS;
Information for table COLOURS
```

```
Columns for table COLOURS:
Column Name                     Data Type        Domain
-----------                     ---------        ------
ENGLISH                         CHAR(8)          MCS_DOM
         DEC_MCS 8 Characters,  8 Octets
FRENCH                          CHAR(8)          MCS_DOM
         DEC_MCS 8 Characters,  8 Octets
JAPANESE                        CHAR(4)          KANJI_DOM
         KANJI 4 Characters,  8 Octets
ROMAJI                          CHAR(8)
         KANJI 8 Characters,  16 Octets
KATAKANA                        CHAR(8)          KATAKANA_DOM
         KATAKANA 8 Characters,  8 Octets
HINDI                           CHAR(8)          HINDI_DOM
         DEVANAGARI 8 Characters,  8 Octets
GREEK                           CHAR(8)          GREEK_DOM
         ISOLATINGREEK 8 Characters,  8 Octets
ARABIC                          CHAR(8)          ARABIC_DOM
         ISOLATINARABIC 8 Characters,  8 Octets
RUSSIAN                         CHAR(8)          RUSSIAN_DOM
         ISOLATINCYRILLIC 8 Characters,  8 Octets

SQL>
```

Example 7: Error displayed if table COLOURS contains data

In the following example, the column ROMAJI is defined with the DEC_KANJI character set. If the column ROMAJI contains data before you alter the character set of the column, SQL displays the following error when you try to retrieve data after altering the table.

```
SQL> SELECT ROMAJI FROM COLOURS;
%RDB-F-CONVERT_ERROR, invalid or unsupported data conversion
-RDMS-E-CSETBADASSIGN, incompatible character sets prohibits the requested
 assignment
SQL> --
SQL> -- To recover, use the ROLLBACK statement or return the column to its
SQL> -- original character set.
SQL> --
SQL> ROLLBACK;
SQL> SELECT ROMAJI FROM COLOURS;
 ROMAJI
 kuro
 shiro
 ao
 aka
 ki
 midori
6 rows selected
SQL>
```

## ATTACH Statement

Specifies the name of a database and the source of the data definitions to be accessed by interactive SQL or by a program at run time. Makes the specified database part of the current database environment. The **database environment** is the set of all databases with unique aliases in the current connection.

The ATTACH statement lets you add new databases at run time; it has no effect on the compile-time environment. To specify the compile-time environment, use the DECLARE ALIAS statement.

You can name either a file or a repository path name to be used for the data definitions.

If a transaction is currently active, SQL returns an informational message and does not attach the specified database environment to the connection.

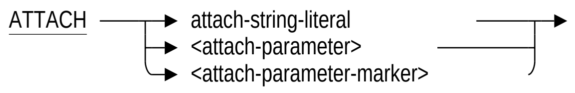
If a database is currently attached and you attach to another database without using an alias, SQL detaches the current database environment and attaches to the specified one in its place.

### Environment

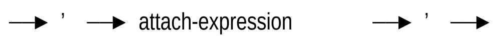
You can use the ATTACH statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

### Format

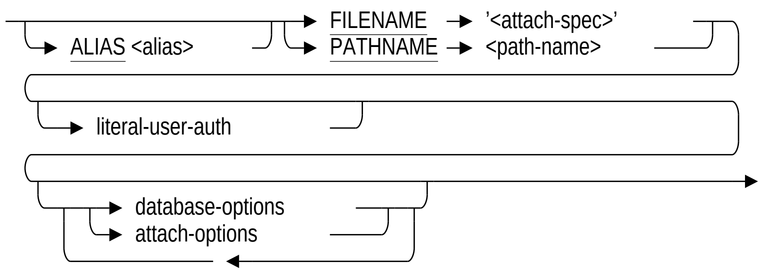


attach-string-literal =

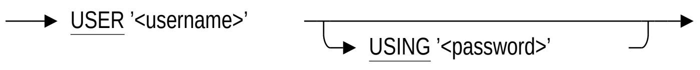


## ATTACH Statement

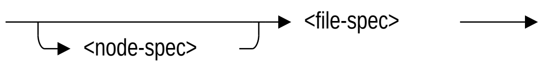
attach-expression =



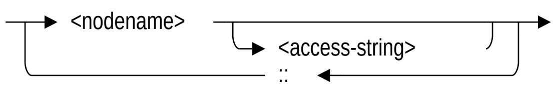
literal-user-auth =



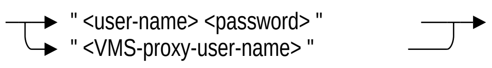
attach-spec =



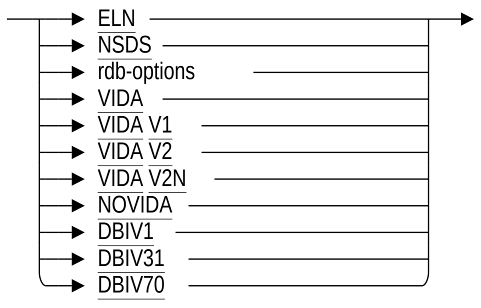
node-spec =



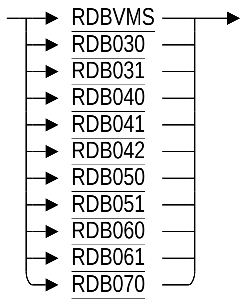
access-string =



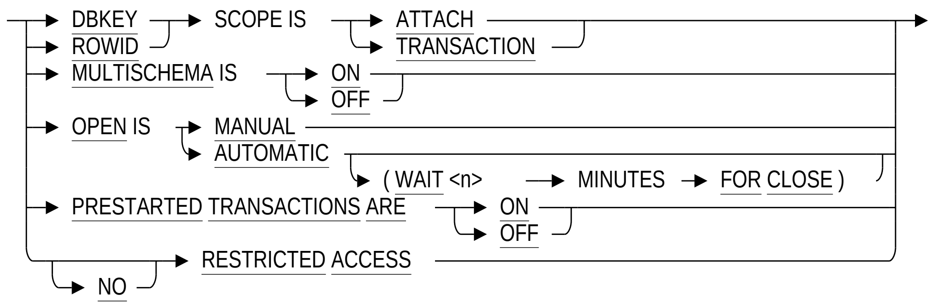
database-options =



rdb-options =



attach-options =



## Arguments

**attach-string-literal**

A character string literal that specifies the database environment for the connection. The attach string literal must contain an attach expression enclosed in single quotation marks.

**attach-parameter**
A host language variable in precompiled SQL or a formal parameter in an SQL module language procedure that specifies the database environment for the connection. The attach parameter must contain an attach expression.

**attach-parameter-marker**
A parameter marker, denoted by question marks (?), in a dynamic SQL statement. The attach parameter marker refers to a parameter that specifies the database environment for the connection. The attach parameter marker must specify a parameter that contains an attach expression.

**attach-expression**
Specifies a database to be added to the environment.

**ALIAS alias**
A part of the attach expression that specifies a name for the attach to the database. Specifying an alias lets your program or interactive SQL statements refer to more than one database.

You do not have to specify an alias in the ATTACH statement. The default alias in interactive SQL and in precompiled programs is RDB$DBHANDLE. In the SQL module language, the default is the alias specified in the module header. Using the default alias (either by specifying it explicitly in the ATTACH statement or by omitting any alias) makes the database part of the default environment. Specifying a **default database** means that statements that refer to that database do not need to use an alias.

If a default alias was already declared, and you specify the default alias in the alias clause (or specify any alias that was already declared), interactive SQL issues an informational message.

In the following example, TESTDB is the first database attached and uses the default alias. When no alias is specified for the second database attached, SQL tries to assign it the default alias but finds that the default alias is already declared.

```
SQL> ATTACH 'FILENAME testdb';
SQL> ATTACH 'FILENAME otherdb';
This alias has already been declared.
Would you like to override this declaration (No)? N
SQL-F-DEFDBDEC, A database has already been declared with the default alias
SQL> SHOW DATABASES;
Default alias:
    Oracle Rdb database in file testdb
SQL> ATTACH 'FILENAME otherdb';
This alias has already been declared.
Would you like to override this declaration (No)? Y
SQL> SHOW DATABASES;
Default alias:
    Oracle Rdb database in file otherdb
```

**FILENAME 'attach-spec'**
A quoted string containing full or partial information needed to access a database.

For an Oracle Rdb database, an attach specification contains the file specification of the .rdb file.

When you use the FILENAME argument, any changes you make to database definitions are entered *only* to the database system file, not to the repository. If you specify FILENAME, your application attaches to the database with that file name at run time.

For information regarding node-spec and file-spec, see Section 2.2.1.1.

OpenVMS VAX OpenVMS Alpha

**PATHNAME path-name**
A full or relative repository path name that specifies the source of the database definitions. When you use the PATHNAME argument, any changes you make to database definitions are entered in both the repository and the database system file. Oracle Rdb recommends using the PATHNAME argument if you have the repository on your system and you plan to use any data definition statements.

If you specify PATHNAME, your application attaches to the database file name extracted from the repository.

The PATHNAME argument can be specified only on OpenVMS platforms. ♦

**literal-user-auth**
Specifies the user name and password to enable access to databases, particularly remote databases

This literal lets you explicitly provide user name and password information in the attach expression.

Digital UNIX

When you use Oracle Rdb for Digital UNIX to attach to a database on a Digital UNIX node, you do not have to explicitly specify the user name and password, even if the database is on a remote Digital UNIX node. Oracle Rdb implicitly authenticates the user whenever the user attaches to a database.

However, you must explicitly provide the user name and password in the following situations:

- If you do not have the same user name and user ID on both nodes
- When you attach to a database on another operating system, such as OpenVMS

You can explicitly provide the user name and password in one of the following ways:

- In SQL statements or command line qualifiers.
- In the configuration file .dbsrc. The following example shows how to include the information in the configuration file:

```
! User name to be used for authentication
SQL_USERNAME   heleng

! Password to be used for authentication
SQL_PASSWORD  MYpassword
```

If you do not specify the USER and USING clause in SQL statements, Oracle Rdb uses the information in the configuration file. ♦

OpenVMS VAX OpenVMS Alpha

When you use Oracle Rdb for OpenVMS to attach to a database in the same cluster, you do not have to explicitly specify the user name and password. Oracle Rdb implicitly authenticates the user whenever the user attaches to a database.

However, when you use Oracle Rdb for OpenVMS to attach to a database on a remote node, even if that node is an OpenVMS node, you must use one of the methods provided by Oracle Rdb to access the database.

You can use one of the following methods to attach to a database on a Digital UNIX node or on a remote OpenVMS node.

- Explicitly provide the user name and password in the ATTACH statement.
- Explicitly provide the user name and password in the configuration file RDB$CLIENT_DEFAULTS.DAT. The following example shows how to include the information in the configuration file:

```
! User name to be used for authentication
SQL_USERNAME   HELENG

! Password to be used for authentication
SQL_PASSWORD  MYPASSWORD
```

You can use one of the following methods to attach to a database on a remote OpenVMS node:

- Use a proxy account on the remote system system.
- Embed the user name and password in the file specification.
- Use the RDB$REMOTE default account.

For information on proxy accounts, embedding the user name in the file specification or using the RDB$REMOTE account, see the *Oracle Rdb7 Guide to SQL Programming*.♦

For more information on configuration files, see the *Migrating Oracle Rdb7 Databases and Applications to Digital UNIX*.

**USER 'username'**
A character string literal that specifies the operating system user name that the database system uses for privilege checking. Because the user name literal is within the quoted attach-string, you must enclose the user name within two sets of single quotation marks in interactive SQL.

This clause also sets the value of the SYSTEM_USER value expression.

**USING 'password'**
A character string literal that specifies the user's password for the user name specified in the USER clause. Because the password literal is within the quoted attach-string, you must enclose surround the password within two sets of single quotation marks in interactive SQL.

**database-options**
By default, the SQL precompiler determines the type of database it attaches to from the type of database specified in compiling the program.

For more information on database options, see Section 2.10.

**attach-options**
Specifies characteristics of the particular database attach. You can specify more than one of these clauses.

**DBKEY SCOPE IS ATTACH**
**DBKEY SCOPE IS TRANSACTION**
Controls when the database key of a deleted record can be used again by SQL. A database key is a unique value that points to a specific table row. There are two options for the DBKEY SCOPE clause:

- The default DBKEY SCOPE IS TRANSACTION means that SQL can reuse the database key of a deleted table row (to refer to a newly inserted row) as soon as the transaction that deleted the original row completes with a COMMIT statement. (If the user who deleted the original row enters a ROLLBACK statement, then the database key for that row cannot be used again by SQL.)

  During the connection of the user who entered the ATTACH statement, the DBKEY SCOPE IS TRANSACTION clause specifies that a database key is guaranteed to refer to the same row *only* within a particular transaction.

  **Note**

  Oracle Rdb recommends using DBKEY SCOPE IS TRANSACTION to reclaim space on a database page faster than if you use DBKEY SCOPE IS ATTACH.

- The DBKEY SCOPE IS ATTACH clause means that SQL cannot use the database key again (to refer to a newly inserted row) until the user who deleted the original row detaches from the database, unless another user is attached using DBKEY SCOPE IS ATTACH. (You detach by declaring another database with the same alias or by using the DISCONNECT statement.)

  During the connection of the user who entered the ATTACH statement, the DBKEY SCOPE IS ATTACH clause specifies that a database key is guaranteed to refer to the same row until the user detaches from the database.

  With the DBKEY SCOPE IS ATTACH clause, a user or program can complete one or several transactions and, while still attached to the database, use database keys (obtained through INSERT, DECLARE CURSOR, FETCH, and singleton SELECT statements) to directly access table rows with less locking and greater speed.

If one user is connected to the database in DBKEY SCOPE IS ATTACH mode, all users are forced to operate in this mode, even if they are are explicitly connected in TRANSACTION mode. That is, no one reuses dbkeys until the ATTACH session disconnects.

See Section 2.6.5 for more information.

**ROWID SCOPE IS ATTACH**
**ROWID SCOPE IS TRANSACTION**
The ROWID keyword is a synonym for the DBKEY keyword. See the DBKEY SCOPE IS argument earlier in this Arguments list for more information.

**MULTISCHEMA IS ON**
**MULTISCHEMA IS OFF**
The MULTISCHEMA IS ON clause enables multischema naming for the duration of the database attach. The MULTISCHEMA IS OFF clause disables multischema naming for the duration of the database attach. On attach, multischema naming defaults to the setting specified during database definition.

You can use multischema naming only when attached to a database that was created with the multischema attribute. If you specify the MULTISCHEMA IS ON clause with a database that was not created with the multischema attribute, SQL returns an error message, as shown in the following example:

```
SQL> ATTACH 'ALIAS PERS_ALIAS FILENAME personnel MULTISCHEMA IS ON';
%SQL-F-NOPHYSMULSCH, The physical multischema attribute was not specified for
the database
```

**PRESTARTED TRANSACTIONS ARE ON**
**PRESTARTED TRANSACTIONS ARE OFF**
Specifies whether Oracle Rdb enables or disables prestarted transactions.

Use the PRESTARTED TRANSACTIONS ARE OFF clause only if your application uses a server process that is attached to the database for long periods of time and causes the snapshot file to grow excessively. If you use the PRESTARTED TRANSACTIONS ARE OFF clause, Oracle Rdb uses additional I/O because each SET TRANSACTION statement must reserve a transaction sequence number (TSN).

For most applications, Oracle Rdb recommends that you enable prestarted transactions. The default is PRESTARTED TRANSACTIONS ARE ON. If you use the PRESTARTED TRANSACTIONS ARE ON clause or do not specify the PRESTARTED TRANSACTIONS clause, the COMMIT or ROLLBACK statement for the previous read/write transaction automatically reserves the TSN for the next transaction and reduces I/O.

You can define the RDMS$BIND_PRESTART_TXN logical name or the RDB_BIND_PRESTART_TXN configuration parameter to define the default setting for prestarted transactions outside of an application. The PRESTARTED TRANSACTION clause overrides this logical name or configuration parameter.

For more information, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**RESTRICTED ACCESS**
**NO RESTRICTED ACCESS**
Restricts access to the database. This allows you to access the database but locks out all other users until you disconnect from the database. Setting restricted access to the database requires DBADM privileges.

The default is NO RESTRICTED ACCESS if not specified.

## Usage Notes

- The ATTACH statement is to be used for dynamic SQL. In precompiled SQL or SQL module language, you must use the DECLARE ALIAS statement to add a database to the implicit environment. For more information, see the DECLARE ALIAS Statement.
- If you attach to the same Oracle Rdb database twice, the SHOW statement may fail with a deadlock error. You can avoid this error by issuing a COMMIT statement. For example:

```
SQL> ATTACH 'FILENAME corporate_data';
SQL> ATTACH 'ALIAS CORP2 FILENAME corporate_data';
SQL> SHOW DATABASES
Default alias:
    Oracle Rdb database in file corporate_data
Alias CORP2:
    Oracle Rdb database in file corporate_data
SQL> SHOW TABLES;
User tables in database with filename corporate_data
     DAILY_HOURS
     DEPARTMENTS
     PAYROLL
     .
     .
     .
     PERSONNEL.WEEKLY_WAGES          A view.
     RECRUITING.CANDIDATES
     RECRUITING.COLLEGES
     RECRUITING.DEGREES
     RECRUITING.RESUMES
```

```
User tables in database with alias CORP2
%RDB-F-DEADLOCK, request failed due to resource deadlock
-RDMS-F-DEADLOCK, deadlock on record 41:413:1
SQL> COMMIT;
SQL> SHOW TABLES;
     User tables in database with filename corporate_data
     DAILY_HOURS
     DEPARTMENTS
     PAYROLL
     .
     .
     .
     User tables in database with alias CORP2
     "CORP2.ADMINISTRATION".ACCOUNTING.DAILY_HOURS
     "CORP2.ADMINISTRATION".ACCOUNTING.DEPARTMENTS
     "CORP2.ADMINISTRATION".ACCOUNTING.PAYROLL
     .
     .
     .
     "CORP2.ADMINISTRATION".RECRUITING.COLLEGES
     "CORP2.ADMINISTRATION".RECRUITING.DEGREES
     "CORP2.ADMINISTRATION".RECRUITING.RESUMES
```

## Examples

Example 1: Attaching a database by file name in interactive SQL and specifying restricted access

This interactive SQL statement attaches the database defined by the file specification mf_personnel to the current connection, and declares the alias pers_alias for that database. Use the SHOW DATABASE statement to see the database settings.

```
SQL> ATTACH 'ALIAS pers_alias FILENAME mf_personnel -
cont> RESTRICTED ACCESS';
```

OpenVMS VAX OpenVMS Alpha

Example 2: Attaching a database by path name in interactive SQL

This interactive SQL statement attaches to the database file name extracted from the repository. Use the SHOW DATABASE statement to see the database settings.

```
SQL> ATTACH
cont> 'ALIAS PERS PATHNAME DISK3:[REPOSITORY.DEPT2]PERSONNEL';
```

♦

Example 3: Using an attach parameter in a program

This excerpt from an SQL module language procedure shows how you might declare a parameter to contain an attach string. You would need to compile the module with the PARAMETER COLONS clause in order to prefix the parameter with a colon.

```
PROCEDURE attach_db
    SQLCODE
    attach_string char(155);

    ATTACH :attach_string;
```

You could then write a C program that calls this procedure. The line that passes the attach string would need a format such as the following:

```
main () {
        long sqlcode;

attach_db( &sqlcode, "ALIAS CORP FILENAME corporate_data" );

/*      Now dynamic statements can refer to alias CORP       */
}
```

Example 4: Explicitly providing the user name and password in the ATTACH statement

The following example shows how to explicitly provide the user name and password in the ATTACH statement. It shows how to attach to an Oracle Rdb for OpenVMS database, from either a Digital UNIX or OpenVMS system.

```
SQL> ATTACH 'FILENAME FARSID::USER1:[GREMBOWSKI.DB]MF_PERSONNEL -
cont>       USER ''grembowski'' USING ''mypassword''';
```

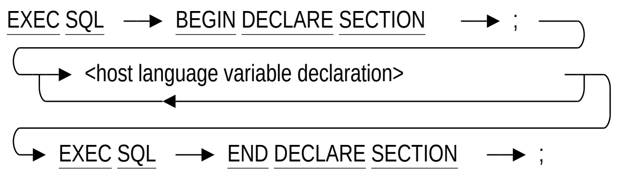
## BEGIN DECLARE Statement

Delimits the beginning of a host language variable declaration section in a precompiled program.

### Environment

You can use the BEGIN DECLARE statement embedded in host language programs to be precompiled.

### Format



### Arguments

**BEGIN DECLARE SECTION**
Delimits the beginning of host language variable declarations.

**; (semicolon)**
Terminates the BEGIN DECLARE and END DECLARE statements.

Which terminator you should use depends on the language in which you are embedding the host language variable. The following table shows which terminator to use.

| | Required SQL Terminator | |
|---|---|---|
| **Host Language** | **BEGIN DECLARE Statement** | **END DECLARE Statement** |
| COBOL | END-EXEC | END-EXEC |
| FORTRAN | None required | None required |
| Ada, C, Pascal, or PL/I | ; (semicolon) | ; (semicolon) |

**host language variable declaration**
A variable declaration embedded in a program.

See Section 2.2.19 for full details on host language variable definitions.

**END DECLARE SECTION**
Delimits the end of host language variable declarations.

## Usage Notes

- The ANSI/ISO SQL standard specifies that host language variables used in embedded SQL statements must be declared within a pair of embedded SQL BEGIN DECLARE . . . END DECLARE statements. If ANSI/ISO SQL compliance is important for your application, you should include all declarations for host language variables used in embedded SQL statements within a BEGIN DECLARE . . . END DECLARE block.
- SQL does not require that you enclose host language variables with BEGIN DECLARE and END DECLARE statements. SQL does, however, issue a warning message if both of the following conditions exist:
  - Your program includes a section delimited by BEGIN DECLARE and END DECLARE statements.
  - You refer to a host language variable that is declared outside the BEGIN DECLARE and END DECLARE block.
- In addition to host language variable declarations, you can include other host language statements within a BEGIN DECLARE . . . END DECLARE block.

## Example

Example 1: Declaring a host language variable within BEGIN . . . END DECLARE statements

The following example shows portions of a Pascal program. The first part of the example declares the host language variable LNAME within the BEGIN DECLARE and END DECLARE statements. The semicolon is necessary as a terminator because the language is Pascal.

The second part of the example shows a singleton SELECT statement that specifies a one-row result table. The statement assigns the value in the row to the previously declared host language variable LNAME.

```
EXEC SQL BEGIN DECLARE SECTION;
  LNAME: packed array [1..20] of char;
EXEC SQL END DECLARE SECTION;
.
.
.
```

```
EXEC SQL
SELECT FIRST_NAME
   INTO :LNAME
   FROM EMPLOYEES
   WHERE EMPLOYEE_ID = "00164";
```

## CALL Statement for Simple Statements

Invokes a stored procedure.

When you define a module with the CREATE MODULE statement, SQL stores the module as an object in an Oracle Rdb database. It also stores each of the module's procedures and functions. The module procedures that reside in an Oracle Rdb database are called **stored procedures**. In contrast, **nonstored procedures** refer to module procedures that reside outside the database in SQL module files. See the CREATE MODULE Statement for more information on creating stored procedures.

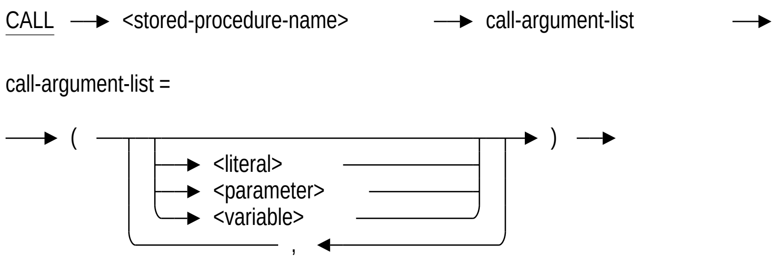
For optional information on invoking stored procedures, see the CALL Statement for Compound Statements.

### Environment

You can use the simple statement CALL:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed
- Inside an external routine

### Format



## Arguments

**procedure-name**
The name of a stored procedure.

**call-argument-list**
Passes a list of literal, parameter values (parameter markers for dynamic execution), or variables to the called stored procedure.

You can pass a literal only to an IN parameter of a stored procedure. You cannot pass a literal to an OUT or INOUT parameter.

In SQL statements to be dynamically executed, you refer to both the main and indicator parameters with a single parameter marker (?). See Section 2.2.19 for details about how to use parameters in programs for static as well as dynamic SQL statement execution.

## Usage Notes

- If the execution of a stored procedure results in an exception, SQL reports the exception as the result of the CALL.
- The number of parameters in the simple statement CALL must match the number of parameters in the procedure that it calls.
- The data types of the parameters used in the simple statement CALL must be equivalent to the data types used in the procedure that it calls.
- Stored and nonstored modules called by the same application cannot have the same name. If you attempt to invoke a stored module while a nonstored module with the same name is active, you receive the following error:

```
%RDB-E-IMP_EXC, facility-specific limit exceeded
-RDMS-E-MODEXTS, there is another module named SALARY_ROUTINES in this
 database
```

## Examples

Example 1: Calling a stored procedure

The following examples show the definition of a stored procedure, NEW_SALARY_PROC, and the nonstored procedure, CALL_NEW_SALARY, that invokes it with the simple statement CALL.

```
SQL> ! The following shows the definition of the stored procedure:
SQL> !
SQL> CREATE MODULE NEW_SALARY_PROC
cont>        LANGUAGE SQL
cont>        PROCEDURE NEW_SALARY_PROC
cont>                  (:ID CHAR (5),
cont>                   :NEW_SALARY INTEGER (2));
cont>         BEGIN
cont>              UPDATE SALARY_HISTORY
cont>                     SET SALARY_END = CURRENT_TIMESTAMP
cont>                     WHERE EMPLOYEE_ID = :ID;
cont>              INSERT INTO SALARY_HISTORY
cont>                                 (EMPLOYEE_ID, SALARY_AMOUNT,
cont>                                  SALARY_START, SALARY_END)
cont>                          VALUES (:ID, :NEW_SALARY,
cont>                                  CURRENT_TIMESTAMP, NULL);
cont>         END;
cont> END MODULE;
SQL>
```

The following example shows an excerpt of an SQL module that contains the nonstored procedure that calls the stored procedure.

```
   .
   .
   .
PROCEDURE CALL_NEW_SALARY
    :ID CHAR(5),
    :ID_IND SMALLINT,
    :NEW_SALARY INTEGER (2),
    :NEW_SALARY_IND SMALLINT,
    SQLCODE;

    CALL NEW_SALARY_PROC (:ID, :NEW_SALARY );
   .
   .
   .
```

Example 2: Calling a procedure in interactive SQL

The following example shows that you use interactive SQL to invoke a stored procedure with the simple statement CALL:

```
SQL> DECLARE :X INTEGER;
SQL>  BEGIN
cont>   SET :X = 0;
cont> END;
SQL> CALL P2 (10, :X);
```

## CALL Statement for Compound Statements

Invokes an external or stored procedure from within a compound statement. That is, invocation must occur with a BEGIN . . . END block.

The OUT and INOUT arguments cannot be general value expressions. They must be variables or parameters. The IN argument can be a general value expression.

When you register a procedure definition with the CREATE PROCEDURE statement, you store information in the database about an external procedure written in a 3GL language. External procedures reside outside the database. The CREATE PROCEDURE statement is documented under the Create Routine Statement. See the Create Routine Statement for more information on creating external procedures.

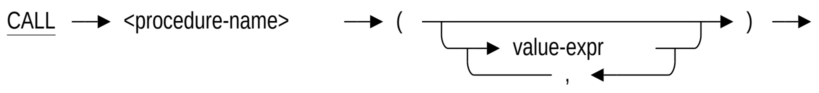
For optional information on invoking stored procedures, see the CALL Statement for Simple Statements.

### Environment

You can use the compound statement CALL:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

### Format



### Arguments

**procedure-name**
The name of the external or stored procedure being invoked.

**value-expr**
Any value expression except DBKEY or aggregate functions. See Section 2.6 for more information on value expressions.

## Usage Notes

- The compound statement CALL can accept, as IN parameters, any value expression. The simple statement CALL is limited to numeric and string literals only and cannot appear within a compound statement.
- The data types of the parameters used in the compound statement CALL must be compatible with the data types used in the procedure that it calls.
- The number of parameters in the compound statement CALL must match the number of parameters in the procedure that it calls.
- The OUT and INOUT parameters must correspond to updatable variables or other OUT and INOUT parameters.
- The values of SQLCODE and SQLSTATE set prior to the compound statement CALL can be examined by the called procedure using the GET DIAGNOSTICS statement. Upon execution of the called procedure, the value in an SQLCODE and SQLSTATE status parameter of the last statement is returned to the caller and can be retrieved using the GET DIAGNOSTICS statement.
- The compound statement CALL can be used within a stored procedure or function to call another stored procedure. When an exception occurs in a nested CALL, that procedure or function and all calling routines return to the topmost caller.
- You cannot call a stored procedure that is in use by the current CALL statement. Recursion is not allowed.

## Examples

Example 1: Calling an external routine within a compound statement

```
BEGIN
   DECLARE :param1 INTEGER;
   CALL extern_routine (:param1, 3);
END;
```

Example 2: Calling a stored procedure from a stored function

```
SQL> CREATE MODULE utility_functions
cont>   LANGUAGE SQL
cont> --
cont>   PROCEDURE trace_date (:dt DATE);
cont>      BEGIN
cont>         TRACE :dt;
cont>      END;
cont> --
cont>   FUNCTION mdy (IN :dt DATE) RETURNS CHAR(10)
cont>   COMMENT ’Returns the date in month/day/year format’;
cont>      BEGIN
cont>         IF :dt IS NULL THEN
cont>            RETURN ’**/**/****’;
cont>         ELSE
cont>            CALL trace_date (:dt);
cont>            RETURN CAST(EXTRACT(MONTH FROM :dt) AS VARCHAR(2)) || ’/’ ||
cont>                   CAST(EXTRACT(DAY FROM :dt) AS VARCHAR(2)) || ’/’ ||
cont>                   CAST(EXTRACT(YEAR FROM :dt) AS VARCHAR(4));
cont>         END IF;
cont>      END;
cont> END MODULE;
```

## CASE Control Statement

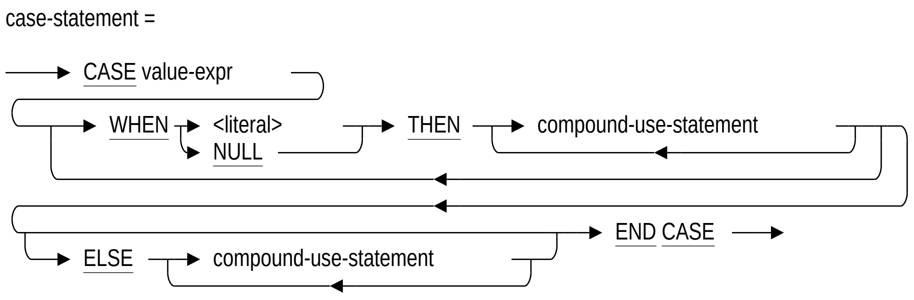
Executes one of a sequence of alternate statement blocks in a compound statement of a multistatement procedure.

### Environment

You can use the CASE control statement in a compound statement of a multistatement procedure:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

### Format



### Arguments

**CASE value-expr**
An expression that evaluates to a single value. SQL compares the CASE clause value expression with each WHEN clause literal value in the WHEN clauses until it finds a match.

The value expression cannot contain a column specification that is not part of a column select expression.

See Section 2.6 for a complete description of the variety of value expressions that SQL provides.

**WHEN literal**
**WHEN NULL**
The literal or NULL value of the WHEN clause that SQL compares with the value expression of the CASE clause. Most CASE control statements include a set of WHEN clauses.

When the values of the WHEN and CASE clauses match, SQL executes the SQL statements associated with that WHEN clause. Control then drops out of the CASE control statement and returns to the next SQL statement after the END CASE clause.

**THEN compound-use-statement**
Executes the set of SQL statements associated with the first WHEN clause in which its argument value matches the CASE value expression.

**ELSE compound-use-statement**
Executes a set of SQL statements when SQL cannot find a WHEN clause that matches the value expression in the CASE clause.

See the Compound Statement for a description of the SQL statements that are valid in a compound statement.

## Usage Notes

- If the CASE value expression cannot find a matching WHEN clause, SQL can take one of the following actions:
  - If an optional ELSE clause is included, SQL executes the set of statements associated with the ELSE clause.
  - If there is no ELSE clause, SQL raises an exception.
- The data type of the CASE value expression and the data type of the WHEN clause literal value must be comparable.
- The literal values of the WHEN clauses in a CASE control statement must be unique. As a corollary, no two WHEN clauses in a CASE control statement can specify a NULL value.

## Examples

Example 1: Using the CASE control statement

```
   .
   .
   .
char x[11];
long x_ind;
   EXEC SQL
      DECLARE ALIAS FOR FILENAME personnel ;

   EXEC SQL
      BEGIN
         CASE :x  INDICATOR :x_ind
            WHEN 'Abrams' THEN
               DELETE FROM employees WHERE . . . ;
            WHEN NULL THEN
               DELETE FROM employees WHERE . . . ;
            ELSE
               DELETE FROM employees WHERE . . . ;
         END CASE ;
      END ;
   .
   .
   .
```

## CLOSE Statement

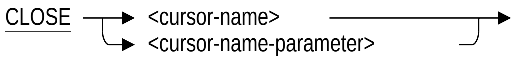
Closes an open cursor.

### Environment

You can use the CLOSE statement:

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

### Format



### Arguments

**cursor-name**
**cursor-name-parameter**
The name of the cursor you want to close. Use a parameter if the cursor referred to by the cursor name was declared at run time with an extended dynamic DECLARE CURSOR statement. Specify the same cursor name parameter used in the dynamic DECLARE CURSOR statement.

You can use a parameter to refer to the cursor name only when the CLOSE statement is accessing a dynamic cursor.

### Usage Notes

- You cannot close a cursor that is not open, or close a cursor that was not named in a DECLARE CURSOR statement.
- If you open a cursor after closing it, SQL positions the cursor before the first row in the result table.
- You can use the SQL CLOSE statement to close cursors individually or use the sql_close_cursors() routine to close all open cursors. The sql_close_cursors() routine takes no arguments. For an example of this routine, see sql_close_cursors.

## Examples

OpenVMS VAX OpenVMS Alpha

Example 1: Closing a cursor declared in a PL/I program

This program fragment uses embedded DECLARE CURSOR, OPEN, and FETCH statements to retrieve and print the name and department of managers. The CLOSE statement closes the cursor after the FETCH statement fails to find any more rows in the result table (when SQLCODE is set to 100).

```
/* Declare the cursor: */
EXEC SQL DECLARE MANAGER CURSOR FOR
        SELECT E.FIRST_NAME, E.LAST_NAME, D.DEPARTMENT_NAME
                FROM EMPLOYEES E, DEPARTMENTS D
                WHERE E.EMPLOYEE_ID = D.MANAGER_ID ;

/* Open the cursor: */
EXEC SQL OPEN MANAGER;

/* Start a loop to process the rows of the cursor: */
DO WHILE (SQLCODE = 0);
        /* Retrieve the rows of the cursor
        and put the value in host language variables: */
        EXEC SQL FETCH MANAGER INTO :FNAME, :LNAME, :DNAME;
        /* Print the values in the variables: */
                        .
                        .
                        .
END;

/* Close the cursor: */
EXEC SQL CLOSE MANAGER;
```
♦

## COMMENT ON Statement

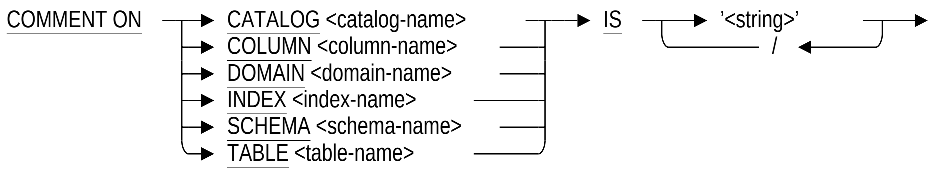
Adds or changes a comment about a catalog, column, domain, index, schema, or table. SQL displays comments on catalogs, columns, schemas, tables, and indexes when you issue a SHOW statement.

### Environment

You can use the COMMENT ON statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

### Format



### Arguments

**CATALOG catalog-name**
Names the catalog for which you want to create a comment. If the catalog is not in the default schema, you must qualify the catalog name in the COMMENT ON statement with an authorization identifier.

**COLUMN column-name**
Names the column for which you want to create a comment. You must qualify the column name with a table name. If the column is not in a table in the default schema, you must qualify the column name in the COMMENT ON statement with both a table name and an authorization identifier.

**DOMAIN domain-name**
Names the domain for which you want to create a comment. If the domain is not in the default schema, you must qualify the domain name in the COMMENT ON statement with an authorization identifier.

**INDEX index-name**
Names the index for which you want to create a comment. If the index is not in the default schema, you must qualify the index name in the COMMENT ON statement with an authorization identifier.

**SCHEMA schema-name**
Names the schema for which you want to create a comment. You must create the schema first. If the schema is not in the default schema, you must qualify the schema name in the COMMENT ON statement with an authorization identifier.

**TABLE table-name**
Names the table for which you want to create a comment. You must create the table definition first. You cannot create comments on views.

**IS 'string'**
Specifies the comment. SQL displays the text when it executes a SHOW statement in interactive SQL. Enclose the comment within single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

## Usage Notes

- You cannot specify the COMMENT ON statement in a CREATE DATABASE statement.

```
SQL> CREATE DATABASE FILENAME TEST
cont> CREATE TABLE TEST_TABLES (COL1 REAL)
cont> COMMENT ON TABLE TEST_TABLES IS 'This will not work';
COMMENT ON TABLE TEST_TABLES IS 'This will not work';
^
%SQL-W-LOOK_FOR_STT, Syntax error, looking for:
%SQL-W-LOOK_FOR_CON,           GRANT, CREATE, ;,
%SQL-F-LOOK_FOR_FIN,    found COMMENT instead
```

- The maximum length for each string literal in a comment is 1,024 characters.

## Example

Example 1: Specifying a comment for columns and tables

```
SQL> -- Change the comment for the WORK_STATUS table:
SQL> COMMENT ON TABLE WORK_STATUS IS
cont> 'Links a status code with 1 of 3 statuses' ;
SQL> SHOW TABLE WORK_STATUS
Information for table WORK_STATUS

Comment on table WORK_STATUS: Links a status code with 1 of 3 statuses
        .
        .
        .
SQL> -- Create a comment for the DEPARTMENT_CODE
SQL> -- column in the DEPARTMENTS table:
SQL> COMMENT ON COLUMN DEPARTMENTS.DEPARTMENT_CODE IS
cont> 'Also used in JOB_HISTORY table';
SQL> SHOW TABLE DEPARTMENTS
Information for table DEPARTMENTS

Comment on table DEPARTMENTS:
information about departments in corporation

Columns for table DEPARTMENTS:
Column Name                     Data Type        Domain
-----------                     ---------        ------
DEPARTMENT_CODE                 CHAR(4)          DEPARTMENT_CODE_DOM
 Comment:       Also used in JOB_HISTORY table
        .
        .
        .
```

Example 2: Specifying a comment containing more than one string literal

```
SQL> COMMENT ON COLUMN EMPLOYEES.EMPLOYEE_ID IS
cont> '1: Used in SALARY_HISTORY table as Foreign Key constraint' /
cont> '2: Used in JOB_HISTORY table as Foreign Key constraint';
SQL> SHOW TABLE (COL) EMPLOYEES;
Information for table EMPLOYEES
```

```
Columns for table EMPLOYEES:
Column Name                     Data Type        Domain
-----------                     ---------        ------
EMPLOYEE_ID                     CHAR(5)          ID_DOM
 Comment:       1: Used in SALARY_HISTORY table as Foreign Key constraint
                2: Used in JOB_HISTORY table as Foreign Key constraint
 Primary Key constraint EMPLOYEES_PRIMARY_EMPLOYEE_ID
LAST_NAME                       CHAR(14)         LAST_NAME_DOM
FIRST_NAME                      CHAR(10)         FIRST_NAME_DOM
MIDDLE_INITIAL                  CHAR(1)          MIDDLE_INITIAL_DOM
ADDRESS_DATA_1                  CHAR(25)         ADDRESS_DATA_1_DOM
ADDRESS_DATA_2                  CHAR(20)         ADDRESS_DATA_2_DOM
CITY                            CHAR(20)         CITY_DOM
STATE                           CHAR(2)          STATE_DOM
POSTAL_CODE                     CHAR(5)          POSTAL_CODE_DOM
SEX                             CHAR(1)          SEX_DOM
BIRTHDAY                        DATE VMS         DATE_DOM
STATUS_CODE                     CHAR(1)          STATUS_CODE_DOM
```

## COMMIT Statement

Ends a transaction and makes permanent any changes that you made during that transaction. The COMMIT statement also:

- Releases all locks
- Closes all open cursors
- Prestarts a new transacation if prestarted transactions are enabled

### Environment

You can use the COMMIT statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

### Format

```
COMMIT WORK
```

### Arguments

**WORK**
An optional keyword that has no effect on the COMMIT statement. The keyword WORK is required by the ANSI/ISO SQL standard. If ANSI/ISO compliance is important for your application, you should include the keyword WORK.

### Usage Notes

- The COMMIT statement affects the following:
  - All databases named in the ON clause of the last DECLARE TRANSACTION or SET TRANSACTION statement plus any databases that were declared since the last DECLARE TRANSACTION or SET TRANSACTION statement. If the last DECLARE TRANSACTION or SET TRANSACTION statement did not include an ON clause, the COMMIT statement affects all declared databases. If the COMMIT

statement is embedded in a program, it affects all the databases declared in the module of the host language program where the transaction was started.

  - All changes made to the data using the DELETE, UPDATE, and INSERT statements.
  - All changes made to the data definitions using the ALTER, CREATE, DROP, GRANT, REVOKE, and COMMENT ON statements.

- In interactive SQL, if you do not issue a COMMIT or ROLLBACK statement before the EXIT statement, SQL returns this message:

  ```
  There are uncommitted changes to this database.
  Would you like a chance to ROLLBACK these changes (No)?
  ```

  The prompt lets you type YES and returns you to interactive SQL. If you type NO or press the Return key, SQL commits the changes made during the last transaction.

  Interactive SQL also has a QUIT statement. The QUIT statement stops an interactive SQL session, rolls back any changes you made, and returns you to the DCL prompt. The QUIT statement does not prompt you for a chance to commit changes.

- In precompiled programs, if your program exits before it issues a COMMIT or ROLLBACK statement, SQL commits the changes if the exit status is successful and rolls them back if it is not. However, Oracle Rdb recommends that you always use an explicit COMMIT or ROLLBACK statement to end a transaction.
- You cannot specify the COMMIT statement in an ATOMIC BEGIN . . . END block.

## Examples

Example 1: Using the COMMIT statement to write a change to the database

This example gives a raise to an employee. To maintain a consistent database, the program performs three operations within one transaction. The program:

- Prompts for an employee identification number (:ID).
- Prompts for a percentage increase, which is used to calculate the raise.
- Uses the UPDATE statement to change the current salary row by changing its salary ending date from null to the current date.

- Uses the INSERT statement to create a new row in the SALARY_HISTORY table. All the columns of the new row can be derived from columns of the old row, except the start date, which must be calculated from the current date. SQL calculates a new value for the SALARY_AMOUNT column from the old record's SALARY_AMOUNT column using the specified percentage increase (:PERC).
- Uses the COMMIT statement to make the changes to the database permanent.

The first two SQL statements in the example are the WHENEVER SQLERROR and WHENEVER SQLWARNING statements. If an error or warning occurs, control transfers to another paragraph that contains a ROLLBACK statement. Therefore, this set of operations is never just partially completed.

```
   .
   .
   .
PROCEDURE DIVISION.
START-UP.

       DISPLAY "Enter employee's ID number:  "
         WITH NO ADVANCING.
       ACCEPT ID.
       DISPLAY "Percentage increase:  "
         WITH NO ADVANCING.
       ACCEPT PERC.

EXEC SQL
          WHENEVER SQLERROR GOTO ERROR-PAR   END_EXEC.

EXEC SQL
          WHENEVER SQLWARNING GOTO ERROR-PAR    END_EXEC.

EXEC SQL SET TRANSACTION READ WRITE RESERVING
             SALARY_HISTORY FOR EXCLUSIVE WRITE
END_EXEC.

EXEC SQL
          UPDATE  SALARY_HISTORY SH
          SET     SH.SALARY_END = CURRENT_TIMESTAMP
          WHERE   SH.EMPLOYEE_ID = :ID
          AND     SH.SALARY_END IS NULL
END_EXEC.
```

```
EXEC SQL
         INSERT INTO SALARY_HISTORY
               (EMPLOYEE_ID, SALARY_AMOUNT, SALARY_START)
               SELECT EMPLOYEE_ID,
                      (SALARY_AMOUNT * (1 + (:PERC / 100))),
                      SALARY_END
               FROM    SALARY_HISTORY
               WHERE   EMPLOYEE_ID = :ID
               AND     CAST(SALARY_END as DATE ANSI) = CURRENT_DATE
END_EXEC.

EXEC SQL
         COMMIT WORK  END_EXEC.
```

Example 2: Using the COMMIT statement with data definition

This example shows a simple database and table definition. The COMMIT statement makes the table definition permanent.

```
SQL> CREATE DATABASE ALIAS INVENTORY;
SQL> --
SQL> CREATE TABLE INVENTORY.PART
cont>    (TEST CHAR(10));
SQL> COMMIT;
SQL> SHOW TABLES
User tables in database with alias INVENTORY
     PART
```

## Compound Statement

Allows you to include more than one SQL statement in an SQL module procedure or in an embedded SQL program. Only by defining a compound statement can you put multiple SQL statements in a procedure. Procedures that contain one or more compound statements are called **multistatement procedures**.

In contrast, a simple statement can contain a single SQL statement only. Procedures that contain a single SQL statement are called **simple-statement procedures**. See the Simple Statement for a description of simple-statement procedures and how you use them in SQL application programming.

A compound statement and a simple statement differ not just in the number of SQL statements they can contain. A compound statement:

- Can include only a subset of the SQL statements allowed in a simple statement procedure. (See the compound-use-statement syntax diagram for a list of these valid statements.)
- Can include control flow statements, much like those you can use in a host language program. (See the control-statement syntax diagrams for a list of flow control statements allowed in a compound statement.)
- Can include transaction management statements, such as ROLLBACK and COMMIT.
- Can include local variables.
- Can control atomicity.
- Can reference only one alias because each compound statement represents a single Oracle Rdb request.

See the *Oracle Rdb7 Guide to SQL Programming* for a conceptual description of compound statements and their relationship to multistatement procedures.

### Environment

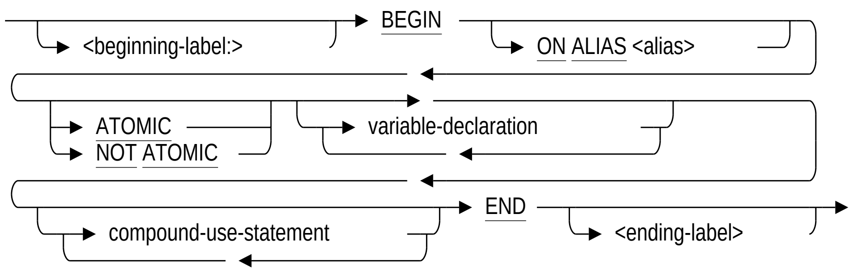
You can use a compound statement:

- In interactive SQL, as a way to test syntax and prototype compound statements for use with programs.
- In embedded SQL, as part of a host language program to be processed with the SQL precompiler.

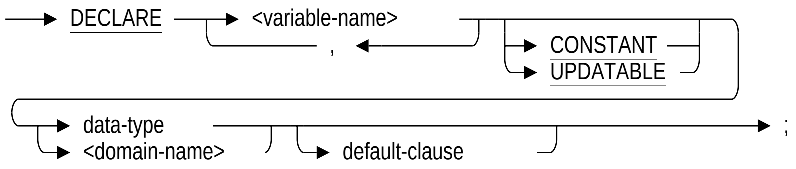
- In SQL module language, as part of a multistatement procedure in an SQL module file to be processed with the SQL module processor.
- In dynamic SQL, to prepare and execute compound statements.

## Format

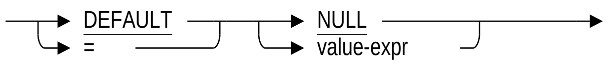
compound-statement =



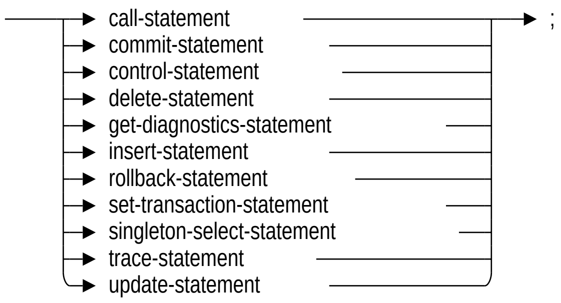
variable-declaration =



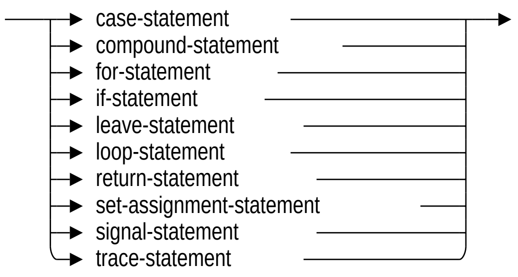
default-clause =



compound-use-statement =



control-statement =



## Arguments

**beginning-label:**
Assigns a name to a block. You use the label with the LEAVE statement to perform a controlled exit from a block or a LOOP statement. Named compound statements are called labeled compound statements. If a block has an ending label, you must also supply an identical beginning label. A label must be unique within the procedure in which the label is contained.

**BEGIN**
Begins a compound statement. The END keyword marks the end of a compound statement. The unit consisting of the BEGIN and END keywords and all statements bounded by them is called a compound statement block or just a **block**. The simplest compound statement block can consist of BEGIN, END, and a terminating semicolon (BEGIN END;).

**ON ALIAS alias**

Specifies an alias allowing your program or interactive SQL statements to refer to more than one database. Use the same alias as specified in the ATTACH statement.

```
SQL> ATTACH 'ALIAS db1 FILENAME mf_personnel';
SQL> ATTACH 'ALIAS db2 FILENAME d1';
SQL> DECLARE :x CHAR(5);
SQL> BEGIN ON ALIAS db1 SELECT EMPLOYEE_ID INTO :x FROM db1.EMPLOYEES
cont> WHERE EMPLOYEE_ID='00164';
cont> END;
SQL> PRINT :x;
 X
 00164
```

**ATOMIC**
**NOT ATOMIC**

Controls whether or not SQL statements in a compound statement are undone when any statement in the compound statement terminates with an exception. Compound statements are NOT ATOMIC by default.

Most single SQL statements are ATOMIC. Only the control statements are NOT ATOMIC. For example, an INSERT statement is ATOMIC, and the entire insert operation either completes or fails as a unit even if it is contained in a NOT ATOMIC block.

- ATOMIC

  In a compound statement defined as ATOMIC, all SQL statements in a compound statement succeed, or when any of the SQL statements in the compound statement raises an exception, they all fail as a unit. Any changes made up to the point of failure are undone. SQL terminates the compound statement as soon as a statement within it fails. SQL does not change variable assignments as a result of a statement failure.

  All statements within an ATOMIC block must be atomic. If you nest compound statements and specify ATOMIC, you must specify ATOMIC for any inner blocks. If you do not, Oracle Rdb returns an error.

- NOT ATOMIC (default)

  In a compound statement defined as NOT ATOMIC, all SQL statements that complete successfully up to the point of a failed statement are not undone as they would be in an ATOMIC compound statement. Partial success of the statements in a NOT ATOMIC compound statement can occur, unlike the all-or-nothing behavior in ATOMIC compound statements. As with ATOMIC compound statements, NOT ATOMIC compound statements are terminated when an SQL statement returns an exception. The partial work of the statement causing a compound statement to terminate is always undone.

SQL restricts the use of SET TRANSACTION, COMMIT, and ROLLBACK statements to NOT ATOMIC compound statements because the nature of an ATOMIC compound statement conflicts with the properties of these statements. The SET TRANSACTION, COMMIT, and ROLLBACK statements cannot be used inside an ATOMIC compound statement even if it is contained in a NOT ATOMIC compound statement. SQL cannot commit a compound statement if a statement should encounter an exception at some point.

**variable-declaration**
Declares local variables for a compound statement. SQL creates variables when it executes a compound statement and deletes them when execution of the compound statement ends.

**CONSTANT**
**UPDATABLE**
CONSTANT changes the variable into a declared constant that cannot be updated. If you specify CONSTANT, you must also have specified the DEFAULT clause to ensure the variable has a value. CONSTANT also indicates that the variable cannot be used as the target of an assignment or be passed as an expression to a procedure's INOUT or OUT parameter.

UPDATABLE is the default and allows the variable to be modified. An update of a variable can occur due to a SET assignment, an INTO assignment (as part of an INSERT, UPDATE, or SELECT statement), an equality (=) comparison, or as a procedure's OUT or INOUT parameter.

**default-clause**
Defines the value of a variable when the statements inside the compound statement begin to execute. You can use any value expression including subqueries, conditional, character, date/time, and numeric expressions as default values. See Section 2.6 for more information about value expressions.

The value expressions described in Section 2.6 include DBKEY and aggregate functions. However, the DEFAULT clause is not a valid location for referencing a DBKEY or an aggregate function. If you attempt to reference either, you receive a compile-time error as shown in the following example:

```
SQL> BEGIN
cont>   DECLARE :x INTEGER DEFAULT DBKEY;
cont> END;
%SQL-F-DBKNOCTX, DBKEY isn’t valid in this context
SQL> --
SQL> BEGIN
cont>   DECLARE :x INTEGER DEFAULT COUNT(*);
cont> END;
%SQL-F-INVFUNREF, Invalid function reference
```

The default can be inherited from the named domain if one exists.

You can also use the equal (=) sign as shown in the following example:

```
SQL> SET FLAGS ’TRACE’;
SQL> BEGIN
cont> DECLARE :x, :y INTEGER DEFAULT -1;
cont> DECLARE :z INTEGER = 3;
cont> TRACE :x, :y, :z;
cont> END;
~Xt: -1         -1         3
```

**compound-use-statement**
Identifies the SQL statements allowed in a compound statement block.

**call-statement**
Invokes an external or stored procedure. See the CALL Statement for Compound Statements for a complete description.

**commit-statement**
Ends a transaction and makes any changes that you made during that transaction permanent. SQL does not allow a COMMIT statement in an ATOMIC compound statement.

See the COMMIT Statement for a complete description.

**control-statement**
The set of statements that provide conditional execution, iterative execution, and cursor-like operations for controlling the execution flow of SQL statements in a compound statement.

**case-statement**
See the CASE Control Statement for a complete description.

**compound-statement**
Lets you nest compound statements in another compound statement.

**for-statement**
See the FOR Control Statement for a complete description.

**if-statement**
See the IF Control Statement for a complete description.

**leave-statement**
See the LEAVE Control Statement for a complete description.

**loop-statement**
See the LOOP Control Statement for a complete.

**return-statement**
Returns the result for stored functions. See the RETURN Control Statement for a complete description.

**set-assignment-statement**
See the SET Control Statement for a complete description.

**signal-statement**
See the SIGNAL Control Statement for a complete description.

**trace-statement**
See the TRACE Control Statement for a complete description.

**delete-statement**
Deletes a row from a table or view.

See the DELETE Statement for a complete description.

**get-diagnostics-statement**
Retrieves diagnostic information for the previously executed statement.

See the GET DIAGNOSTICS Statement for a complete description.

**insert-statement**
Adds a new row, or a number of rows, to a table or view. For compound statements, SQL restricts the INSERT statement to database insert operations in a single database.

See the INSERT Statement for a complete description.

**rollback-statement**
Ends a transaction and undoes all changes you made since that transaction began. SQL does not allow a ROLLBACK statement in an ATOMIC compound statement.

See the ROLLBACK Statement for a complete description.

**set-transaction-statement**
Starts a transaction and specifies its characteristics.

See the SET TRANSACTION Statement for a complete description.

**singleton-select-statement**
Specifies a one-row result table.

See the SELECT Statement: Singleton Select for a complete description.

**trace-statement**
Writes values to the trace log file. See the TRACE Control Statement for a complete description.

**update-statement**
Modifies a row in a table or view.

See the UPDATE Statement for a complete description.

**END**
Ends a compound statement block.

**ending-label**
Assigns a name to a block. If a block has a beginning label, you must use the same name for the ending label.

## Usage Notes

- In a compound statement, variable declarations must appear before any executable SQL statement. For example, SQL returns an error if you put the SET statement before any DECLARE statement.

```
SQL> BEGIN
cont> DECLARE :mgrid CHAR(5);
cont> DECLARE :cur_mgrid CHAR(5);
cont> SET :mgrid = ’00167’;
cont> DECLARE :state_code CHAR(2);
%SQL-I-DEPR_FEATURE, Deprecated Feature: Keyword DECLARE used as an
identifier
DECLARE :state_code CHAR(2);
         ^
%SQL-W-LOOK_FOR_STT, Syntax error, looking for:
%SQL-W-LOOK_FOR_CON,           FOR, LOOP, BEGIN, WHILE,
%SQL-F-LOOK_FOR_FIN,    found STATE_CODE instead
```

- In interactive SQL and precompiled SQL, you cannot use a label on the outermost compound statement. You can use labels on compound statements nested in another compound statement.

  In SQL module language, you can put a label on the outermost compound statement.

- Use the BEGIN ON ALIAS syntax to specify the database to which a compound statement refers. If you do not use BEGIN ON ALIAS, the following error is returned:

```
SQL> ATTACH 'ALIAS db1 FILENAME mf_personnel';
SQL> ATTACH 'ALIAS db2 FILENAME d1';
SQL> DECLARE :x CHAR(5);
SQL> BEGIN
cont> SELECT EMPLOYEE_ID INTO :x FROM db1.EMPLOYEES
cont> WHERE EMPLOYEE_ID='00164';
cont> END;
%SQL-F-ONEDBINMOD, Only one alias is legal in this module
```

- A compound statement can reference only one alias because each compound statement represents a single Oracle Rdb request.
- You cannot refer to more than one database in a multistatement procedure.
- The compound-use statements are executed sequentially.

  If any statement raises an exception, all database work is undone. If the failed statement is inside an ATOMIC block, all work of this block is undone. The procedure that contains the statement ends with the exception reported through the SQLCODE, SQLSTATE, or the SQLCA parameter.

  If all statements execute, the compound statement executes.

- A new timestamp is calculated for every statement in a NOT ATOMIC compound statement. Alternatively, a new timestamp is calculated only once for an ATOMIC compound statement. Consider using ATOMIC statements for complex multistatement procedures to reduce CPU overhead.
- The LIST OF BYTE VARYING data type is not permitted as the explicit type of a variable or domain used for the type.
- The default value is assigned before any other executable statements in the compound block. The default value cannot reference the variables being declared by the current DECLARE clause. The default value can reference variables in outer blocks or other complex value expressions.

- If the DEFAULT clause is not present, the default of the domain, if one exists, is used to initialize the variable. Otherwise, the declared variables initial value is undefined.
- If a list of variables are declared together, the DEFAULT is applied to each variable. This is shown in the following example which displays the default values using the TRACE statement:

  ```
  SQL> SET FLAGS 'TRACE';
  SQL> BEGIN
  cont>    DECLARE :x, :y INTEGER DEFAULT -1;
  cont>    TRACE :x, :y;
  cont> END;
  ~Xt: -1          -1
  ```

- The default clause is reassigned whenever the variable declaration re-enters scope. For example, if the DECLARE clause appears in a loop, the variable is re-initialized on each iteration of the loop.
- A FOR cursor loop executes the DO . . . END FOR body of the loop for each row fetched from the row set. Applications cannot use RETURNED_SQLCODE or RETURNED_SQLSTATE to determine if the FOR loop reached the end of the row set without processing any rows. Applications should use the GET DIAGNOSTICS ROW_COUNT statement after the END FOR clause to test for zero or more rows processed.

## Examples

Example 1: Using a compound statement to update rows

The following compound statement uses variables to update rows in the JOBS table. It uses the SET asssignment control statement to assign a value to the variable MIN_SAL.

```
SQL> BEGIN
cont> -- Declare the variable.
cont>      DECLARE :MIN_SAL INTEGER(2);
cont> -- Set the value of the variable.
cont>      SET :MIN_SAL = (SELECT MIN(MINIMUM_SALARY) FROM JOBS) * 1.08;
cont> -- Update the rows in the JOBS table.
cont>      UPDATE JOBS
cont>            SET MINIMUM_SALARY = :MIN_SAL
cont>                WHERE MINIMUM_SALARY < (:MIN_SAL * 1.08);
cont> END;
```

Example 2: Using the DEFAULT clause

The following example shows several variable declarations using a variety of value expressions for the DEFAULT clause.

```
SQL> SET FLAGS 'TRACE';
SQL>
SQL> BEGIN
cont>   DECLARE :x INTEGER DEFAULT -1;
cont>   TRACE :x;
cont> END;
~Xt: -1
SQL>
SQL> BEGIN
cont>   DECLARE :x INTEGER DEFAULT NULL;
cont>   TRACE COALESCE (:x, 'NULL');
cont> END;
~Xt: NULL
SQL>
SQL> BEGIN
cont>   DECLARE :x INTEGER DEFAULT (1+1);
cont>   TRACE :x;
cont> END;
~Xt: 2
SQL>
SQL> BEGIN
cont>   DECLARE :x INTEGER DEFAULT (SELECT COUNT(*) FROM EMPLOYEES);
cont>   TRACE :x;
cont> END;
~Xt: 100
```

Example 3: Specifying a LOOP statement using the DEFAULT clause

The following example shows some simple value expressions. The default value is applied to :y on each iteration of the loop, not just the first time the statement is executed.

```
SQL> BEGIN
cont>    DECLARE :x INTEGER DEFAULT 0;
cont>    WHILE :x < 10
cont>    LOOP
cont>        BEGIN
cont>           DECLARE :y INTEGER DEFAULT 1;
cont>           TRACE :x, :y;
cont>           SET :x = :x + :y;
cont>           SET :y = :y + 1;
cont>        END;
cont>    END LOOP;
cont> END;
~Xt: 0         1
~Xt: 1         1
~Xt: 2         1
~Xt: 3         1
~Xt: 4         1
~Xt: 5         1
~Xt: 6         1
~Xt: 7         1
~Xt: 8         1
~Xt: 9         1
```

## CONNECT Statement

Creates a database environment and a connection, and specifies a connection name for that association.

A **connection** specifies an association between the set of cursors, intermediate result tables, and procedures in all modules of an application and the database environment currently attached.

A database environment is one or more databases that can be attached or detached as a unit. The **connection name** designates a particular connection and database environment. When you execute a procedure, it executes in the context of a connection.

When you issue a CONNECT statement, SQL creates a new connection from all the procedures in your application and creates a new environment from all the databases named in the CONNECT statement. The new environment can include databases already attached in the default environment.

There are two ways to attach a database to the default environment:

- Use an ATTACH statement to specify a database environment at run time. All the databases you specify with subsequent ATTACH statements become part of the default environment.
- Use a DECLARE ALIAS statement to specify a database environment at compile time in precompiled SQL and SQL module language. All the databases that you specify using DECLARE ALIAS statements also become part of the default environment.

A CONNECT statement creates a new connection with a new set of attachments, and does an implicit SET CONNECT to that new connection. Although a CONNECT statement does not create a transaction, each connection has its own implicit transaction context. You can issue two different CONNECT statements that attach to the same database, but each attach is unique.

Once you have specified a connection name in a CONNECT statement, you can refer to that connection name in subsequent SET CONNECT statements. You can use a SET CONNECT statement to specify a new connection for an application to run against without having to detach and recompile queries. See the SET CONNECT Statement for more information.

The DISCONNECT statement detaches from databases, ends the transactions in the connections that you specify, and rolls back all the changes you made since those transactions began.

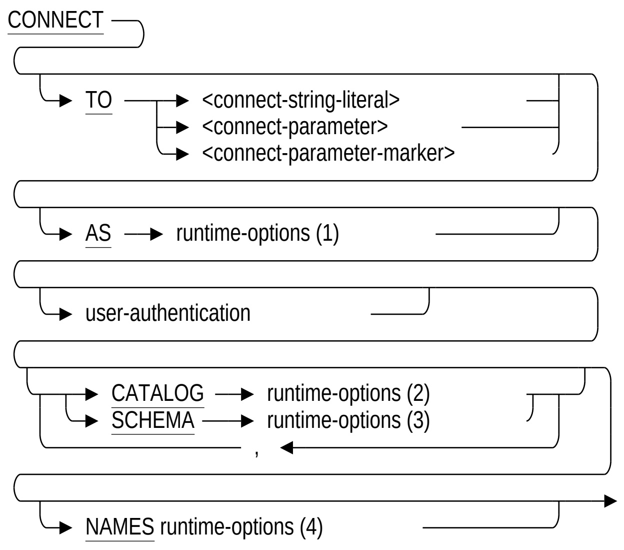
## CONNECT Statement

### Environment

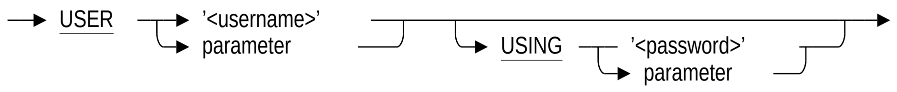
You can use the CONNECT statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

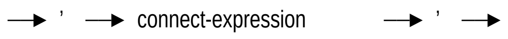
### Format



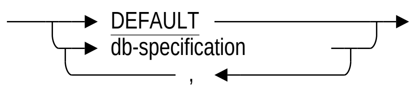
user-authentication =



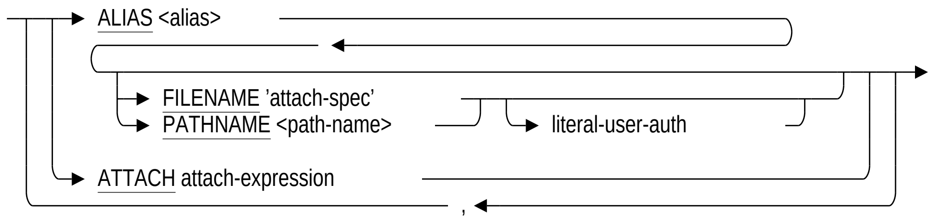
connect-string-literal =



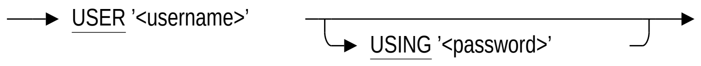
connect-expression =



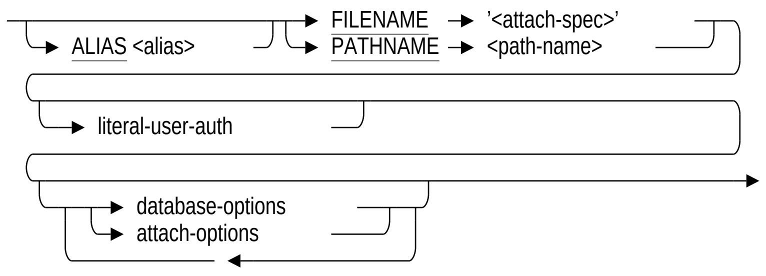
db-specification =



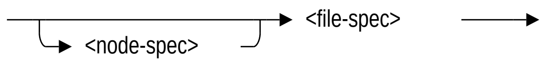
literal-user-auth =



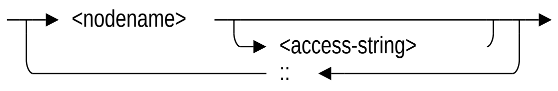
attach-expression =



attach-spec =

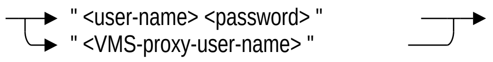


node-spec =

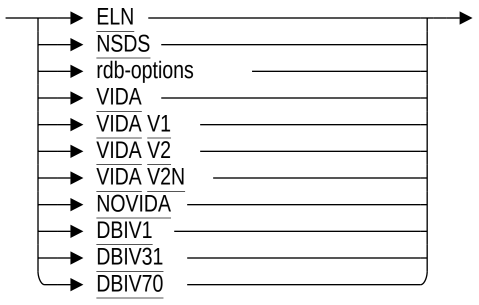


## CONNECT Statement

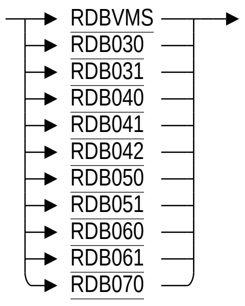
access-string =



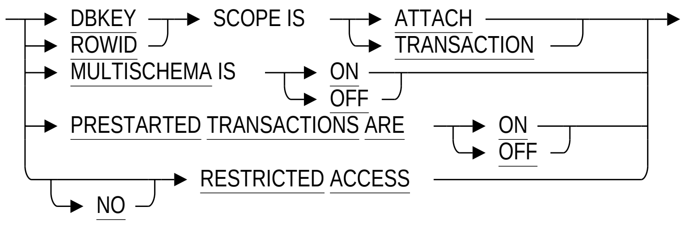
database-options =



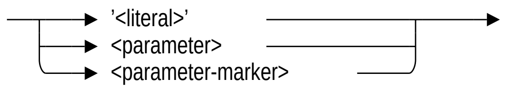
rdb-options =



connect-options =



runtime-options



## Arguments

**TO connect-string-literal**
**TO connect-parameter**
**TO connect-parameter-marker**
Specifies the database environment. You can supply a parameter marker from dynamic SQL, a host language variable from a precompiled SQL program, a parameter from an SQL module language module, or a string literal. The argument that you supply must be a character string that contains a connect expression that is interpreted at run time.

**db-specification**
Specifies one or more valid aliases. An alias, which identifies a particular database, is valid only if that database is either declared in any of the modules in the current application or attached with the ATTACH statement. You can issue an ATTACH statement as part of the db-specification.

**ALIAS alias**
Specifies a name for a particular attach to a database. Specifying an alias in the connect expression lets your program or interactive SQL statements refer to more than one database.

You do not have to specify an alias in the CONNECT statement if you are referring only to the default database.

If you specify an alias, but do not specify a FILENAME or PATHNAME, SQL uses the path name or file name in the DECLARE ALIAS statement for that database by default. The alias must be part of the default environment. The PATHNAME argument is available only on OpenVMS platforms.

**literal-user-auth**
Specifies the user name and password for the specified alias in the connection. This clause enables access to databases, particularly remote databases.

This literal lets you explicitly provide user name and password information for each alias in the CONNECT statement. For more information about when to use this clause, see the ATTACH Statement.

**USER 'username'**
A character string literal that specifies the operating system user name that the database system uses for privilege checking.

**USING 'password'**
A character string literal that specifies the user's password for the user name specified in the USER clause.

**FILENAME 'attach-spec'**
A quoted string containing full or partial information needed to access a database.

For an Oracle Rdb database, an attach specification contains the file specification of the .rdb file.

When you use the FILENAME argument, any changes you make to database definitions are entered *only* to the database system file, not to the repository. If you specify FILENAME, your application attaches to the database with that file name at run time.

For information regarding node-spec and file-spec, see Section 2.2.1.1.

OpenVMS VAX OpenVMS Alpha

**PATHNAME path-name**
A full or relative repository path name that specifies the source of the schema definitions. When you use the PATHNAME argument, any changes you make to schema definitions are entered in the repository and the database system file. Oracle Rdb recommends using the PATHNAME argument if you have the repository on your system and you plan to use any data definition statements.

The path name that you specify overrides the path name associated with the alias at run time.

If you specify PATHNAME at run time, your application attaches to the database file name extracted from the repository. ♦

**user-authentication**
Specifies the user name and password to enable access to databases, particularly remote databases.

This clause lets you explicitly provide user name and password information in the CONNECT statement. If you do not specify user name and password information in the ALIAS clause or the ATTACH clause, SQL uses the user name and password specified in this clause as the default for each alias specified.

For more information about when to use this clause, see the ATTACH Statement.

**USER 'username'**
**USER parameter**
A character string literal that specifies the operating system user name that the database system uses for privilege checking.

**USING 'password'**
**USING parameter**
A character string literal that specifies the user's password for the user name specified in the USER clause.

**ATTACH attach-expression**
Specifies an alias that is not part of the default environment. See the ATTACH Statement for details about the FILENAME 'attach-spec', PATHNAME path-name, database-options, and attach-options.

**AS runtime-options (1)**
Specifies an identifier for the association between the group of databases being attached (the environment) and the database and request handles that reference them (the connection).

The connection name must be unique within your application. Use a literal string enclosed within single quotation marks, for example:

```
CONNECT TO 'ALIAS CORP FILENAME corporate_data' AS 'JULY_CORP_DATA'
```

If you do not specify a connection name, SQL generates a unique connection name. For example:

```
SQL> CONNECT TO
cont> 'ATTACH FILENAME mf_personnel';
SQL> SHOW CONNECTIONS
        RDB$DEFAULT_CONNECTION
->      SQL$CONN_00000000
```

**'literal'**
**parameter**
**parameter-marker**
Specifies a character set name that is used as the default, identifier, and literal character sets for the session of the current connection. The value of runtime-options must be one of the character sets listed in Table 2-3.

**CATALOG runtime-options (2)**
Specifies the default catalog for dynamic statements in the connection.

You can supply a parameter marker from dynamic SQL, a host language variable from a precompiled SQL program, a parameter from an SQL module language module, or a string literal. The argument that you supply must be a character string that contains a connect expression that is interpreted at run time.

**SCHEMA runtime-options (3)**
Specifies the schema for dynamic statements in the connection.

You can supply a parameter marker from dynamic SQL, a host language variable from a precompiled SQL program, a parameter from an SQL module language module, or a string literal. The argument that you supply must be a character string that contains a connect expression that is interpreted at run time.

**NAMES runtime-options (4)**
Specifies the name for dynamic statements in the connection. See Section 2.1 for more detail.

You can supply a parameter marker from dynamic SQL, a host language variable from a precompiled SQL program, a parameter from an SQL module language module, or a string literal. The argument that you supply must be a character string that contains a connect expression that is interpreted at run time.

## Usage Notes

- If you specify a list of aliases, SQL uses this as the run-time parameters for the database with the matching alias.
- When you issue the CONNECT statement, the default environment is determined by the global and local database of the module containing the CONNECT statement. If a database is declared as LOCAL, the module has its own view of the database environment. When the application calls procedures in modules with local aliases, the database environment changes. If you name the same local alias in two different modules, SQL considers this two different databases.

  If a database is declared as GLOBAL, SQL shares the database between modules. If you declare all aliases as GLOBAL, the default connection does not change. If you name an alias declared as GLOBAL in two different modules, SQL shares the database between modules.

- You must declare a database as GLOBAL to reference the database name in CONNECT statements that are in different modules from the DECLARE statement for the database.
- To enable connections, use the CONNECT qualifier on the module language command line, or the SQLOPTIONS=(CONNECT) qualifier on the precompiler command line. When you enable connections, dynamic SQL statements can access all global databases, and the CONNECT statement can connect to any of the global databases.
- If your application calls a procedure that has no currently active connection, SQL uses the default environment. The default environment at that point is formed by all databases declared using the DECLARE ALIAS statement in that module. Databases in other modules are attached when procedures in that module are executed (assuming that no transaction is active).
- The DISCONNECT statement ends active transactions and undoes all changes to the databases during that attach. To incorporate changes, you must issue a COMMIT statement before issuing a DISCONNECT statement.
- You can use the SET CONNECT statement to select a connection from the available connections.
- OpenVMS VAX OpenVMS Alpha You can use SQL connections and explicit calls to DECdtm system services to control when you attach and detach from specific databases. By explicitly calling DECdtm system services and associating each database with an SQL connection, you can detach from one database while remaining attached to other databases. For more information, see the *Oracle Rdb7 Guide to Distributed Transactions*. ♦
- The specified character set must contain ASCII characters. See Table 2-3 for a list of allowable character sets. The option can be a literal, a parameter, or a parameter marker.
- SQL uses DEC_MCS if the NAMES clause is not specified. Each connection specifies a new character set.
- The character set also specifies the character set for the SQLNAME field in SQLDA and SQLDA2 for statements without an explicit database context.
- OpenVMS VAX OpenVMS Alpha If the database default character set is not DEC_MCS, the PATHNAME specifier cannot be used due to a current limitation of the CDD/Repository, where object names must only contain DEC_MCS characters. SQL flags this as an error. ♦

## Examples

Example 1: Creating a default connection and one other connection

The following example shows how a user attaches to one database with two different connections: the default connection and the named connection TEST.

```
SQL> ATTACH 'ALIAS MIA1 FILENAME mia_char_set';
SQL> CONNECT TO 'ALIAS MIA1 FILENAME mia_char_set' AS 'TEST';
SQL> SHOW CONNECTIONS;
        -default-
->      TEST
SQL> SHOW CONNECTIONS TEST;
Connection: TEST
Default alias is RDB$DBHANDLE
Default catalog name is RDB$CATALOG
Default schema name is DAY
Dialect: SQLV40
Default character unit: OCTETS
Keyword Rules: SQLV40
View Rules: SQLV40
Default DATE type: DATE VMS
Quoting Rules: SQLV40
Optimization Level: DEFAULT
Default character set is DEC_MCS
National character set is DEC_MCS
Identifier character set is DEC_MCS
Literal character set is DEC_MCS

Alias MIA1:
        Identifier character set is DEC_KANJI
        Default character set is DEC_KANJI
        National character set is KANJI
```

```
SQL> --
SQL> -- The following example shows how to specify the NAMES
SQL> -- clause in the CONNECT statement that changes the session
SQL> -- default, identifier, and literal character sets.
SQL> --
SQL> CONNECT TO 'ALIAS MIA1 FILENAME mia_char_set' AS 'TEST1'
cont> NAMES 'DEC_KANJI';
SQL> SHOW CONNECTIONS;
        -default-
        TEST
->      TEST1
SQL> SHOW CONNECTIONS CURRENT
Connection: TEST1
Default alias is RDB$DBHANDLE
Default catalog name is RDB$CATALOG
Default schema name is DAY
Dialect: SQLV40
Default character unit: OCTETS
Keyword Rules: SQLV40
View Rules: SQLV40
Default DATE type: DATE VMS
Quoting Rules: SQLV40
Optimization Level: DEFAULT
Default character set is DEC_KANJI
National character set is DEC_MCS
Identifier character set is DEC_KANJI
Literal character set is DEC_KANJI

Alias MIA1:
        Identifier character set is DEC_KANJI
        Default character set is DEC_KANJI
        National character set is KANJI
SQL> SHOW CHARACTER SET;
Default character set is DEC_KANJI
National character set is DEC_MCS
Identifier character set is DEC_KANJI
Literal character set is DEC_KANJI

Alias MIA1:
        Identifier character set is DEC_KANJI
        Default character set is DEC_KANJI
        National character set is KANJI
```

Example 2: Creating a default connection and two other connections

The following example attaches to three databases: personnel_northwest, personnel_northeast, and personnel_southeast. (By not specifying an alias for personnel_northwest, it is assigned the default alias.) Several connections are established, including EAST_COAST, which includes both personnel_northeast and personnel_southeast.

Use the SHOW DATABASE statement to see the changes to the database.

```
SQL> --
SQL> -- Attach to the personnel_northwest and personnel_northeast databases.
SQL> -- Personnel_northwest has the default alias, so personnel_northeast
SQL> -- requires an alias.
SQL> -- All of the attached databases comprise the default connection.
SQL> --
SQL> ATTACH 'FILENAME personnel_northwest';
SQL> ATTACH 'ALIAS NORTHEAST FILENAME personnel_northeast';
SQL> --
SQL> -- Add the personnel_southeast database.
SQL> --
SQL> ATTACH 'ALIAS SOUTHEAST FILENAME personnel_southeast';
SQL> --
SQL> -- Connect to personnel_southeast.  CONNECT does an
SQL> -- implicit SET CONNECT to the newly created connection.
SQL> --
SQL> CONNECT TO 'ALIAS SOUTHEAST FILENAME personnel_southeast'
cont>     AS 'SOUTHEAST_CONNECTION';
SQL> --
SQL> -- Connect to both personnel_southeast and personnel_northeast as
SQL> -- EAST_COAST connection. SQL replaces the current connection to
SQL> -- the personnel_southeast database with the EAST_COAST connection
SQL> -- when you issue the CONNECT statement. You now have two different
SQL> -- connections that include personnel_southeast.
SQL> --
SQL> CONNECT TO 'ALIAS NORTHEAST FILENAME personnel_northeast,
cont>     ALIAS SOUTHEAST FILENAME personnel_southeast'
cont>     AS 'EAST_COAST';
SQL> --
SQL> -- The DEFAULT connection still includes all of the attached databases.
SQL> --
SQL> SET CONNECT DEFAULT;
SQL> --
SQL> -- DISCONNECT releases the connection name EAST_COAST, but
SQL> -- does not detach from the EAST_COAST databases because
SQL> -- they are also part of the default connection.
SQL> --
SQL> DISCONNECT 'EAST_COAST';
SQL> --
SQL> SET CONNECT 'EAST_COAST';
%SQL-F-NOSUCHCON, There is not an active connection by that name
```

```
SQL> --
SQL> -- If you disconnect from the default connection, and have no other
SQL> -- current connections, you are longer be attached to any databases.
SQL> --
SQL> DISCONNECT DEFAULT;
SQL> SHOW DATABASES;
%SQL-F-ERRATTDEF, Could not use databae file specified by SQL$DATABASE
-RDB-E-BAD_DB_FORMAT, SQL$DATABASE does not reference a database known to Rdb
-RMS-E-FNF, file not found
```

## CREATE CACHE Clause

**Note**

You cannot issue CREATE CACHE as an independent statement. It is a clause allowed only as part of a CREATE DATABASE or IMPORT statement.

You can also create a row cache area using the ADD CACHE clause of the ALTER DATABASE statement.

---

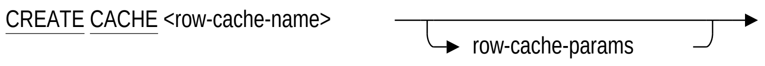
Creates a row cache area that allows frequently referenced rows to remain in memory even when the associated page has been transferred back to disk. This saves in memory usage because only the more recently referenced rows are cached versus caching the entire buffer.

See the ALTER DATABASE Statement and the CREATE DATABASE Statement for more information regarding the row cache areas.

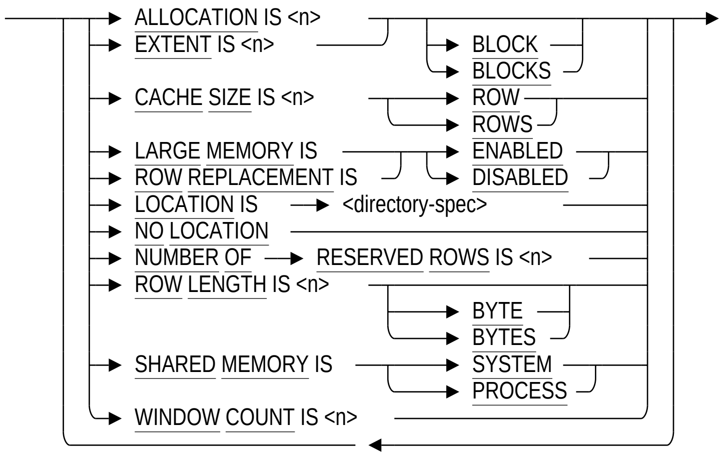
### Environment

You can use the CREATE CACHE clause only within a CREATE DATABASE or IMPORT statement.

### Format



row-cache-params =



## Arguments

**CACHE row-cache-name**
Creates a row cache.

**ALLOCATION IS n BLOCK**
**ALLOCATION IS n BLOCKS**
Specifies the initial allocation of the row cache file (.rdc) to which cached rows are written.

If the ALLOCATION clause is not specified, the default allocation in blocks is approximately 40% of the CACHE SIZE for this cache.

**EXTENT IS n BLOCK**
**EXTENT IS n BLOCKS**
Specifies the file extent size for the row cache file (.rdc).

If the EXTENT clause is not specified, the default number of blocks is CACHE SIZE * 127 for this cache.

**CACHE SIZE IS n ROW**
**CACHE SIZE IS n ROWS**
Specifies the number of rows allocated to the row cache area. As the row cache area fills, rows more recently referenced are retained in the row cache area while those not referenced recently are discarded. Adjusting the allocation of

the row cache area helps to retain important rows in memory. If not specified, the default is 1000 rows.

### LARGE MEMORY IS ENABLED
### LARGE MEMORY IS DISABLED

OpenVMS Alpha

Specifies whether or not large memory is used to manage the row cache. **Very large memory (VLM)** allows Oracle Rdb to use as much physical memory as is available and to dynamically map it to the virtual address space of database users. It provides access to a large amount of physical memory through small virtual address windows.

Use LARGE MEMORY IS ENABLED only when both of the following are true:

- You have enabled row caching.
- You want to cache large amounts of data, but the row cache area does not fit in the virtual address space.

The default is DISABLED. See the Usage Notes for restrictions pertaining to the very large memory (VLM) feature. ♦

### ROW REPLACEMENT IS ENABLED
### ROW REPLACEMENT IS DISABLED

Specifies whether or not Oracle Rdb replaces rows in the cache. When ROW REPLACEMENT IS ENABLED, rows are replaced when the row cache area becomes full. When ROW REPLACEMENT IS DISABLED, rows are not replaced when the row cache area is full. The type of row replacement policy depends upon the application requirements for each cache.

The default is ENABLED.

### LOCATION IS directory-spec

Specifies the name of the directory to which row cache information is written. The database system generates a file name (row-cache-name.rdc) automatically for each row cache area at checkpoint time. Specify a device name and directory name only, enclosed within single quotation marks. The file name is the row-cache-name specified when creating the row cache area. By default, the location is the directory of the database root file. These .rdc files are permanent database files.

This LOCATION clause overrides a previously specified location at the database level.

### NO LOCATION

Removes the location previously specified in a LOCATION IS clause for the row cache area. If you specify NO LOCATION, the row cache location becomes the directory of the database root file.

**NUMBER OF RESERVED ROWS IS n**

Specifies the maximum number of cache rows that each user can reserve. The default is 20 rows.

**ROW LENGTH IS n BYTE**
**ROW LENGTH IS n BYTES**

Specifies the length of each row allocated to the row cache area. Rows are not cached if they are too large for the row cache area. The ROW LENGTH is an aligned longword rounded up to the next multiple of 4 bytes.

The maximum row length in a row cache area is 65535 bytes.

If the ROW LENGTH clause is not specified, the default row length is 256 bytes.

**SHARED MEMORY IS SYSTEM**
**SHARED MEMORY IS PROCESS**

OpenVMS Alpha

Determines whether cache global sections are created in system space or process space. The default is SHARED MEMORY IS PROCESS.

When you use cache global sections created in the process space, you and other users share physical memory and the OpenVMS Alpha operating system maps a row cache area to a private address space for each user. As a result, all users are limited by the free virtual address range and each use a percentage of memory in overhead. If many users are accessing the database, the overhead can be high.

When many users are accessing the database, consider using the SHARED MEMORY IS SYSTEM clause. This gives users more physical memory because they share the system space of memory and there is none of the overhead associated with the process space of memory. ♦

**WINDOW COUNT IS n**

OpenVMS Alpha

Specifies the number of virtual address windows used by the LARGE MEMORY clause.

The window is a view into the physical memory used to create the very large memory (VLM) information. Because the VLM size may be larger than that which can be addressed by a 32-bit pointer, you need to view the VLM information through small virtual address windows.

You can specify a positive integer in the range from 10 through 65535. The default is 100 windows. ♦

## Usage Notes

- If the name of the row cache area is the same as any logical area (for example a table name, index name, storage map name, RDB$SEGMENTED_STRINGS, RDB$SYSTEM_RECORD, and so forth), then this is a logical area cache and the named logical area is cached automatically. Otherwise, a storage area needs to be associated with the cache.
- The CREATE CACHE clause does not assign the row cache area to a storage area. You must use the CACHE USING clause with the CREATE STORAGE AREA clause of the CREATE DATABASE statement or the CACHE USING clause with the ADD STORAGE AREA or ALTER STORAGE AREA clauses of the ALTER DATABASE statement.
- The product of the CACHE SIZE and the ROW LENGTH settings determines the amount of memory required for the row cache area (some additional overhead and rounding up to page boundaries is performed by the database system).
- The row cache area is shared by all processes attached to the database on any node.
- The following are requirements when using the row caching feature:
  - After-image journaling must be enabled
  - Fast commit must be enabled
  - Number of cluster notes must equal 1
- Use the SHOW CACHE statement to view information about a cache.

## Examples

Example 1: Creating a row cache area

This example creates a database, creates a row cache area, and assigns the row cache area to a storage area.

```
SQL> CREATE DATABASE FILENAME test_db
cont> ROW CACHE IS ENABLED
cont> CREATE CACHE test1
cont>    CACHE SIZE IS 100 ROWS
cont> CREATE STORAGE AREA area1
cont>    CACHE USING test1;
SQL> SHOW CACHE
Cache Objects in database with filename test_db
        TEST1
SQL> SHOW CACHE test1

     TEST1
        Cache Size:            100 rows
        Row Length:            256 bytes
        Row Replacement:       Enabled
        Shared Memory:         Process
        Large Memory:          Disabled
        Window Count:          100
        Reserved Rows:         20
        Sweep Rows:            3000
        No Sweep Thresholds
        Allocation:            100 blocks
        Extent:                100 blocks
SQL> SHOW STORAGE AREA area1

     AREA1
         Access is:      Read write
         Page Format:    Uniform
         Page Size:      2 blocks
         Area File:      SQL_USER1:[DAY.V70]AREA1.RDA;1
         Area Allocation:          402 pages
         Area Extent Minimum:      99 pages
         Area Extent Maximum:      9999 pages
         Area Extent Percent:      20 percent
         Snapshot File:  SQL_USER1:[DAY.V70]AREA1.SNP;1
         Snapshot Allocation:      100 pages
         Snapshot Extent Minimum:  99 pages
         Snapshot Extent Maximum:  9999 pages
         Snapshot Extent Percent:  20 percent
         Extent :       Enabled
         Locking is Row Level
         Using Cache TEST1
No database objects use Storage Area AREA1
```

## CREATE CATALOG Statement

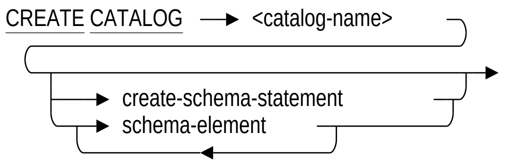
Creates a name for a group of schemas in a multischema database.

### Environment

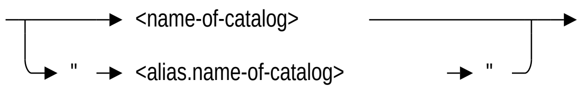
You can use the CREATE CATALOG statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

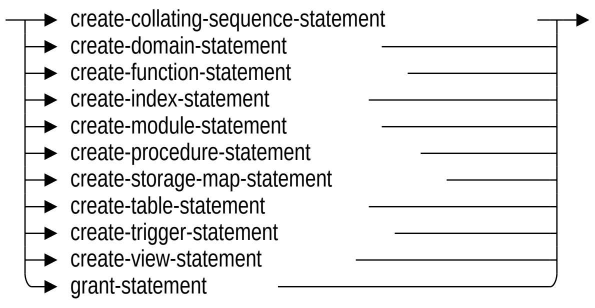
### Format



catalog-name =



schema-element =



## Arguments

**catalog-name**
The name of the catalog definition you want to create. Use any valid SQL name that is unique among all catalog names in the database. For more information on catalog names, see Section 2.2.7.

**"alias.name-of-catalog"**
Specifies an optional name for the attach to the database. Always qualify the catalog name with an alias if your program or your interactive SQL statements refer to more than one database. Separate the name of the catalog from the alias with a period, and enclose the qualified name within double quotation marks.

**create-schema-statement**
For more information, see the CREATE SCHEMA Statement.

**schema-element**
One or more CREATE statements or a GRANT statement. For more information, see the CREATE SCHEMA Statement.

## Usage Notes

- You can create a catalog only in a database that has the multischema attribute. Use the MULTISCHEMA IS ON clause in the CREATE DATABASE or ALTER DATABASE statement to give a database the multischema attribute.
- Even if a database has the multischema attribute, you cannot create a catalog in that database if multischema naming is disabled. Multischema naming is enabled by default for databases with the multischema attribute, but you can disable it using the MULTISCHEMA IS OFF clause of the DECLARE ALIAS or ATTACH statement.
- If you attached to a database using an alias, you must use the same alias to specify elements in that database in subsequent statements. When you qualify a catalog name with an alias, you must separate the alias from the catalog name with a period and enclose the name pair within double quotation marks.

  Before issuing a statement with a qualified catalog name, you must issue a SET QUOTING RULES statement, specify a QUOTING RULES clause in a DECLARE MODULE statement embedded in a program, or specify a QUOTING RULES clause in an SQL module file.

- If you set the ANSI/ISO SQL standard flagger on, the CREATE CATALOG statement is flagged as nonstandard syntax.
- SQL stores schemas and schema elements in RDB$CATALOG if you do not change the default catalog.
- The name of the catalog created in CREATE CATALOG is the default catalog for the whole statement.

## Examples

Example 1: Creating a catalog for a database using an alias

This example shows how an interactive user could attach to the sample database called personnel and create a catalog in that database. (You must use the personnel sample database created with the multischema attribute for this example.) Using an alias, the user distinguishes the personnel database from other databases that may be attached later in the same session.

```
SQL> ATTACH 'ALIAS CORPORATE FILENAME personnel -
cont> MULTISCHEMA IS ON';
SQL> --
SQL> -- SQL creates a default catalog called RDB$CATALOG in
SQL> -- each multischema database.
SQL> --
SQL> SHOW CATALOG;
Catalogs in database personnel
    "CORPORATE.RDB$CATALOG"
SQL> --
SQL> -- The SET QUOTING RULES 'SQL92' statement allows the use of
SQL> -- double quotation marks, which SQL requires when you
SQL> -- qualify a catalog name with an alias.
SQL> --
SQL> SET QUOTING RULES 'SQL92';
SQL> CREATE CATALOG "CORPORATE.MARKETING";
SQL> --
SQL> SHOW CATALOG;
Catalogs in database personnel
    "CORPORATE.MARKETING"
    "CORPORATE.RDB$CATALOG"
```

Example 2: Creating a catalog in the database with the default alias

This example shows a CREATE CATALOG clause used in an interactive CREATE DATABASE statement. In this example, the user creates a database without specifying an alias. Because the user is not attached to any other databases, the new database becomes the default alias.

```
SQL> CREATE DATABASE FILENAME inventory
cont> MULTISCHEMA IS ON
cont> CREATE CATALOG PARTS
cont> CREATE SCHEMA PRINTERS AUTHORIZATION DAVIS
cont> CREATE TABLE LASER EXTERNAL NAME IS DEPT_2_LASER
cont> (SERIAL_NO INT, LOCATION CHAR)
cont> CREATE SCHEMA TERMINALS AUTHORIZATION DAVIS
cont> CREATE TABLE TERM100 EXTERNAL NAME IS DEPT_2_TERM100
cont> (SERIAL_NO INT, LOCATION CHAR);
SQL> SHOW CATALOG;
Catalogs in database with filename inventory
    PARTS
    RDB$CATALOG
SQL> show schemas;
Schemas in database with filename inventory
    PARTS.PRINTERS
    PARTS.TERMINALS
    RDB$SCHEMA
```

## CREATE COLLATING SEQUENCE Statement

OpenVMS VAX OpenVMS Alpha

Identifies a collating sequence that has been defined using the OpenVMS National Character Set (NCS) utility. Use the CREATE COLLATING SEQUENCE statement to identify collating sequences other than the database default collating sequence that you plan to use with certain domains. (The default collating sequence for a database is established by the COLLATING SEQUENCE IS clause in the CREATE SCHEMA statement; if you omit that clause at database definition time, the default sequence is ASCII.)

You must enter a CREATE COLLATING SEQUENCE statement specifying a collating sequence before you enter the name of that sequence in any of the following statements:

- CREATE DOMAIN . . . COLLATING SEQUENCE
- CREATE DOMAIN . . . NO COLLATING SEQUENCE
- ALTER DOMAIN . . . COLLATING SEQUENCE
- ALTER DOMAIN . . . NO COLLATING SEQUENCE

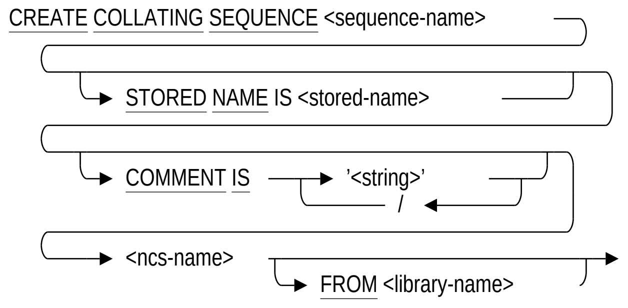
This statement can be used only on OpenVMS platforms.

### Environment

You can use the CREATE COLLATING SEQUENCE statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

## Format


CREATE COLLATING SEQUENCE <sequence-name>
STORED NAME IS <stored-name>
COMMENT IS
'<string>'
/
<ncs-name>
FROM <library-name>

## Arguments

**sequence-name**
Specifies the name by which the collating sequence named in the ncs-name argument is known to the schema. The ncs-name and sequence-name arguments can be the same.

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access a collating sequence created in a multischema database. The stored name allows you to access multischema definitions using interfaces, such as Oracle RMU, the Oracle Rdb management utility, that do not recognize multiple schemas in one database. You cannot specify a stored name for a collating sequence in a database that does not allow multiple schemas.

**COMMENT IS 'string'**
Adds a comment about the collating sequence. SQL displays the text when it executes a SHOW COLLATING SEQUENCE statement in interactive SQL. Enclose the comment within single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

**ncs-name**
Specifies the name of a collating sequence in the default NCS library, SYS$LIBRARY:NCS$LIBRARY, or in the NCS library specified by the library-name argument.

The collating sequence can be either one of the predefined NCS collating sequences or one that you defined yourself using NCS.

**FROM library-name**
Specifies the name of an NCS library other than the default. The default NCS library is SYS$LIBRARY:NCS$LIBRARY.

## Usage Notes

- The CREATE COLLATING SEQUENCE statement is the first step in specifying an alternate collating sequence for a domain. After you create the collating sequence, you can apply it to a particular domain.
- The following list shows abbreviated forms of all the statements that involve collating sequences. You must define your collating sequence using the CREATE COLLATING SEQUENCE statement before you enter a CREATE DOMAIN . . . COLLATING SEQUENCE or ALTER DOMAIN . . . COLLATING SEQUENCE statement.
  - CREATE DOMAIN . . . COLLATING SEQUENCE sequence-name;
  - CREATE DOMAIN . . . NO COLLATING SEQUENCE;
  - ALTER DOMAIN . . . COLLATING SEQUENCE sequence-name;
  - ALTER DOMAIN . . . NO COLLATING SEQUENCE;
  - DROP COLLATING SEQUENCE sequence-name;
  - CREATE SCHEMA . . . create-collating-sequence-statement;
  - CREATE SCHEMA . . . COLLATING SEQUENCE sequence-name;
  - IMPORT . . . COLLATING SEQUENCE sequence-name;
  - SHOW . . . COLLATING SEQUENCE;
- You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a read/write transaction implicitly.
- Other users are allowed to be attached to the database when you issue the CREATE COLLATING SEQUENCE statement.
- You cannot execute the CREATE COLLATING SEQUENCE statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. For more information on the RDB$SYSTEM storage area, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

- If you attempt to define a database with the following collating sequence, a bugcheck dump results with an exception at RDMS$$MCS$NCS_RECODE_8 + 00000665.

  ```
  native_2_upper_lower = cs(
  sequence = (%X00,"#"," ","A","a","B","b","C","c","D","d","E",
  "e","8","F","f","5"-"4","G","g","H","h","I","i","J","j","K","k",
  "L","l","M","m","N","n","9","O","o","1","P","p","Q","q","R","r",
  "S","s","7"-"6","T","t","3"-"2","U","u","V","v","W","w","X","x",
  "Y","y","Z","z"),
  modifications = (%X01-%X1F=%X00,"!"-""""=%X00,"$"-"0"=%X00,":"-"@"=
  %X00,
  "{"-%XFF=%X00,""="A"));
  ```

  The modifications portion of the collating sequence results in too many characters being converted to NULL. Oracle Rdb can handle converting only about 80 characters to NULL.

  A workaround is to modify the MULTINATIONAL2 character set to sort in the desired order.

- You cannot use any of the following as a collating sequence name:
  - "MCS"
  - "ASCII"
  - "        " (all spaces)
  - Null character (a special character whose character code is 0)

- The maximum length for each string literal in a comment is 1,024 characters.

- Because of some special characteristics of the Norwegian collating sequence, certain restrictions apply when creating a Norwegian collating sequence in a database. The name of a Norwegian collating sequence in the NCS library must begin with the character string NORWEGIAN.

  Please note that the sequence customarily shipped with OpenVMS is named NORWEGIAN which meets this restriction. You may wish to alter the Norwegian sequence slightly or change its name. Oracle recommends that any variation of the Norwegian collating sequence be given a name such as NORWEGIAN_1 or NORWEGIANA.

- Oracle Rdb for Digital UNIX does not support the creation of collating sequences with SQL. You can, however, restore a database from OpenVMS and retain the collating sequences that exist in that database. Also, if a Digital UNIX database is altered from an OpenVMS system, then collating sequences can be created remotely.

## Example

Example 1: Creating a French collating sequence

The following example creates a collating sequence using the predefined collating sequence FRENCH. It then shows the defined collating sequence by using the SHOW COLLATING SEQUENCE statement.

```
SQL> CREATE COLLATING SEQUENCE FRENCH FRENCH;
SQL> --
SQL> SHOW COLLATING SEQUENCE
User collating sequences in schema with filename SQL$DATABASE
     FRENCH
```

Example 2: Create a Spanish collating sequence specifying more than one comment

```
SQL> CREATE COLLATING SEQUENCE SPANISH_COL
cont> COMMENT IS 'first comment' / 'second comment'
cont> SPANISH;
SQL> SHOW COLLATING SEQUENCE SPANISH_COL;
     SPANISH_COL
 Comment:       first comment
                second comment
```

♦

## CREATE DATABASE Statement

Creates database system files, metadata definitions, and user data that comprise a database. The CREATE DATABASE statement lets you specify in a single SQL statement all data and privilege definitions for a new database. (You can also add definitions to the database later.) For information about ways to ensure good performance and data consistency, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

The many optional elements of the CREATE DATABASE statement make it very flexible. In its simplest form, the CREATE DATABASE statement creates database system files, specifies their names, and determines the physical characteristics of the database. Using the optional elements of the CREATE DATABASE statement, you can also specify:

- Whether the database created with CREATE DATABASE is **multifile** (separate database root file and storage area data file) or **single file** (combined database root file and storage area data file). Multifile databases can have many storage areas for user data, all separate from the database root file created by the CREATE DATABASE statement. Multifile databases include CREATE STORAGE AREA clauses in the CREATE DATABASE statement to create multiple storage area files for enhanced performance.

  The presence or absence of a CREATE STORAGE AREA clause in a CREATE DATABASE statement determines whether the database is single file or multifile. To create a multifile database, you must include a CREATE STORAGE AREA clause in the CREATE DATABASE statement. To create a single-file database, do not include a CREATE STORAGE AREA clause in the CREATE DATABASE statement.

- Values for various database root file parameters that override the system defaults. **Database root file** (.rdb) parameters describe characteristics of the database root file. Database root file parameters affect the entire database, whether it is a single-file or a multifile database.

- Values for storage area parameters that override system defaults. **Storage area** parameters describe characteristics of the database storage area files. In a single-file database, because the storage area data file is combined with the database root file, storage area parameters apply to a single storage area and affect the entire database. In a multifile database, storage area parameters specify defaults for the main storage area, RDB$SYSTEM, and for any subsequent CREATE STORAGE AREA clauses within the CREATE DATABASE statement.

- Any number of database elements. Database elements are a CREATE CATALOG statement, a CREATE STORAGE AREA clause, or a GRANT statement. The CREATE DATABASE statements that create single-file databases cannot include a CREATE STORAGE AREA clause because this is specific to multifile databases. The CREATE DATABASE statements that create multifile databases must include at least one CREATE STORAGE AREA clause.

  Unlike the same statements outside a CREATE DATABASE statement, database elements do not use statement terminators. The first statement terminator that SQL encounters ends the CREATE DATABASE statement. Later CREATE or GRANT statements are not within the scope of the CREATE DATABASE statement.

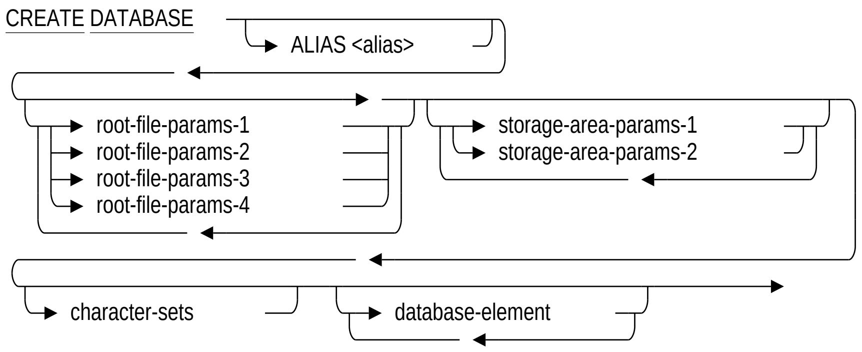
- The database default character set and national character set. For information regarding identifier character sets, database default character sets, and national character sets, see Section 2.1.2, Section 2.1.3, and Section 2.1.4, respectively.

## Environment

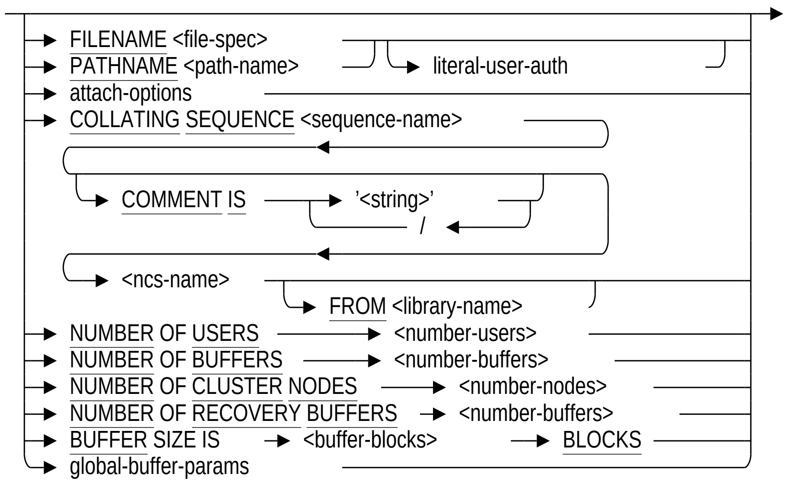
You can use the CREATE DATABASE statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

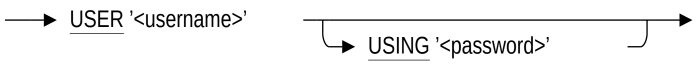
## Format



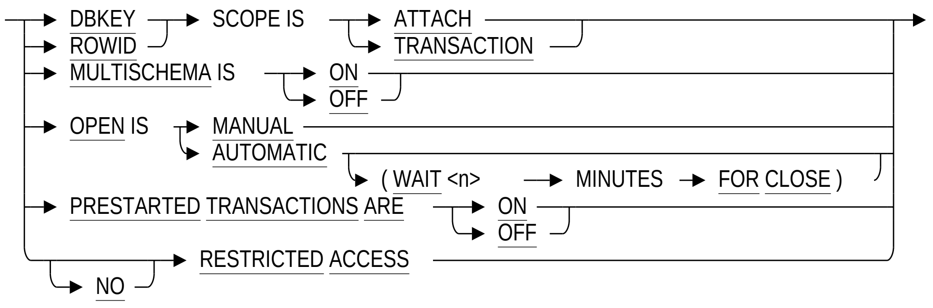
root-file-params-1 =



literal-user-auth =

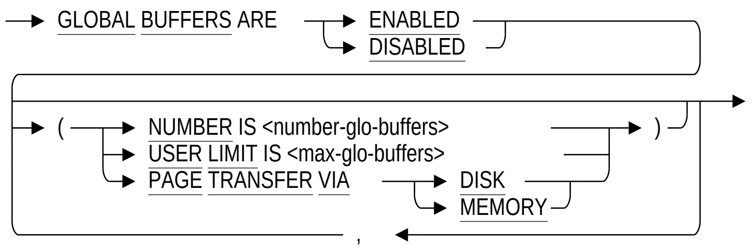


attach-options =

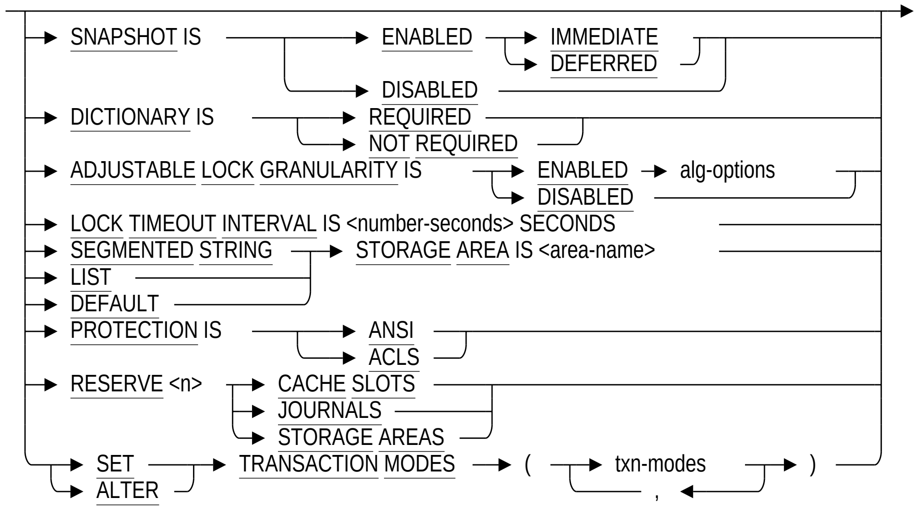


## CREATE DATABASE Statement

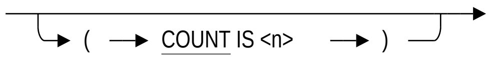
global-buffer-params=



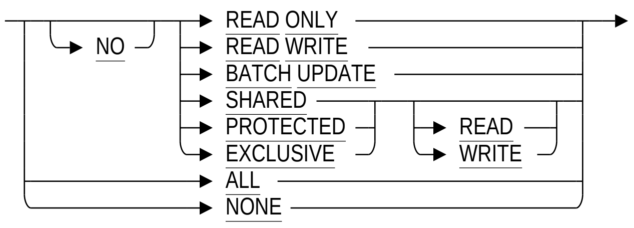
root-file-params-2 =



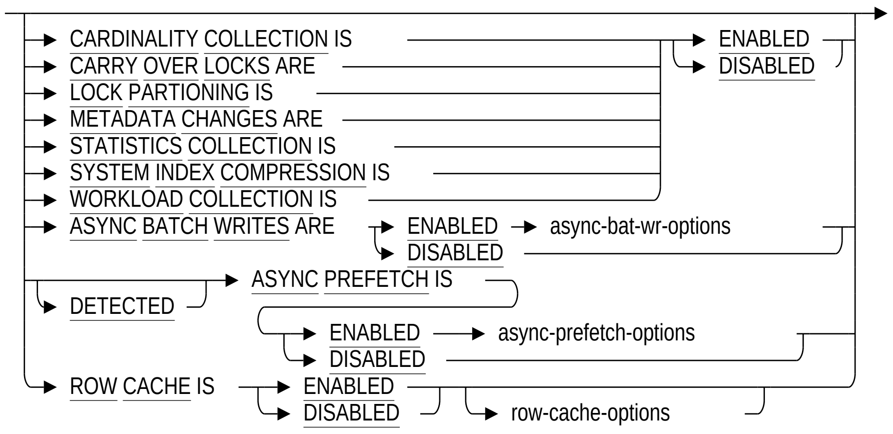
alg-options =



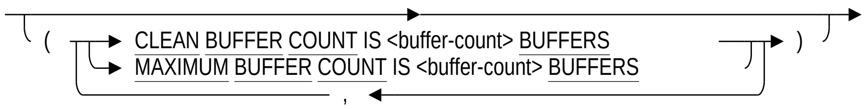
txn-modes =



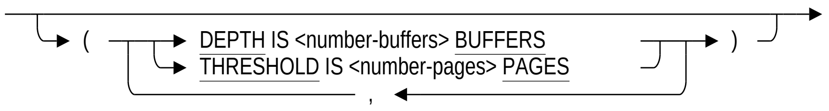
root-file-params-3 =



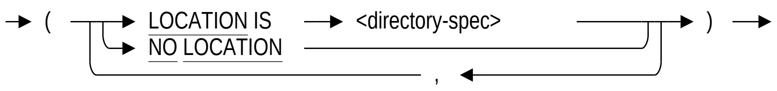
asynch-bat-wr-options =


(
CLEAN BUFFER COUNT IS <buffer-count> BUFFERS
MAXIMUM BUFFER COUNT IS <buffer-count> BUFFERS
)
,

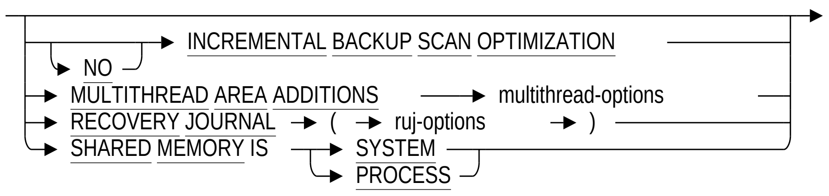
async-prefetch-options =


(
DEPTH IS <number-buffers> BUFFERS
THRESHOLD IS <number-pages> PAGES
)
,

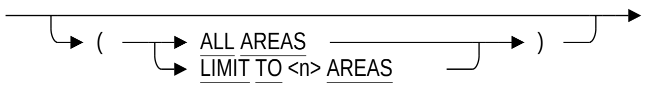
row-cache-options =



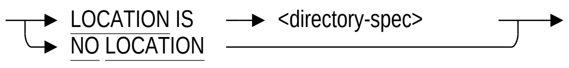
root-file-params-4 =



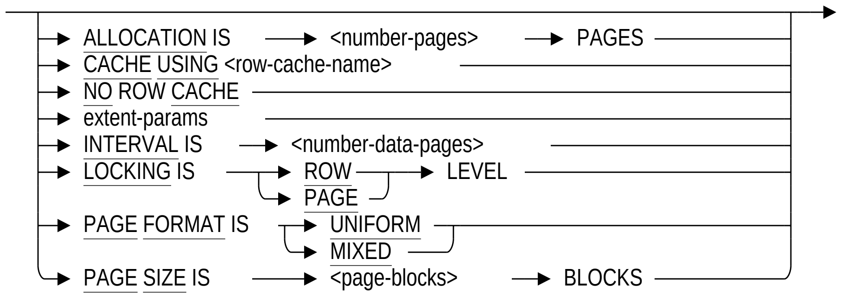
multithread-options =



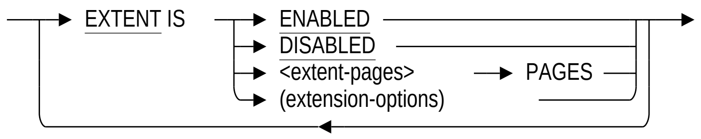
ruj-options =



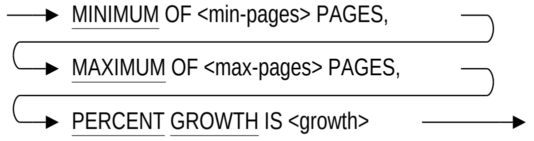
storage-area-params-1 =



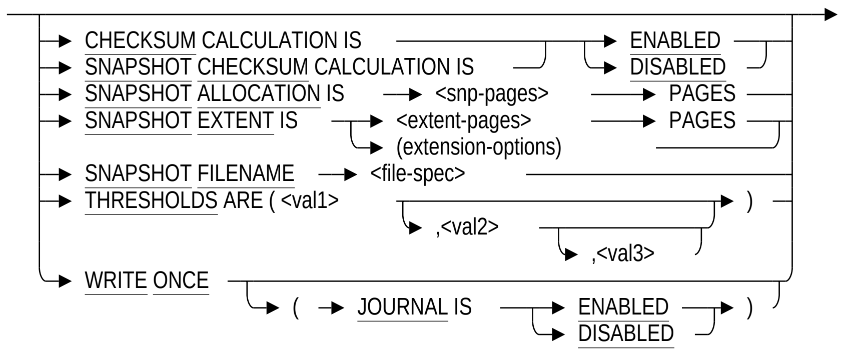
extent-params =



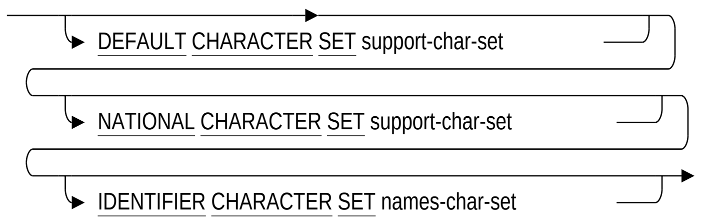
extension-options =


MINIMUM OF <min-pages> PAGES,
MAXIMUM OF <max-pages> PAGES,
PERCENT GROWTH IS <growth>

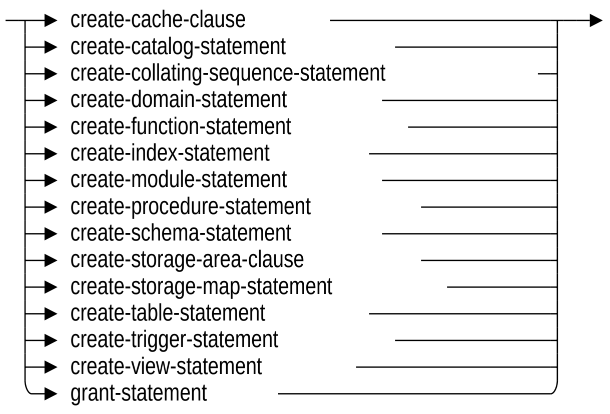
storage-area-params-2 =



character-sets =


DEFAULT CHARACTER SET support-char-set
NATIONAL CHARACTER SET support-char-set
IDENTIFIER CHARACTER SET names-char-set

database-element =



## Arguments

**alias**

Specifies the alias for the implicit database declaration executed by the CREATE DATABASE statement. An **alias** is a name for a particular attach to a database that identifies that database in subsequent SQL statements.

**Note**

If you attach to a database using an alias, you must use that alias in subsequent statements to qualify the names of elements in that database.

If you omit the FILENAME argument from the database root file parameters, SQL also uses the alias as the file name for the database root file and creates the root file in the current default directory. (SQL generates a syntax error if you include a disk or directory specification in the alias clause.) You must specify either the FILENAME or alias argument.

Schema elements in the CREATE DATABASE statement do not need to use the alias, however, they cannot specify any other alias.

The alias clause is optional. The default alias in interactive SQL and in precompiled programs is RDB$DBHANDLE. In the SQL module language, the default is the alias specified in the module header. Using the default alias (either by specifying it explicitly in the ALIAS clause or omitting the ALIAS clause) declares the database as the default database. Specifying a default database means that statements outside the CREATE DATABASE statement that refer to the default database do not need to use an alias.

If a default database was already declared, and you specify the default alias in the ALIAS clause (or specify any alias that was already declared), the results depend on the environment in which you issue the CREATE DATABASE statement.

- In interactive SQL, you receive a prompt asking if you want to override the default database declaration. Unless you explicitly override the default declaration, the CREATE DATABASE statement fails.

  ```
  SQL> -- Assume a default database has been declared:
  SQL> --
  SQL> -- Now create a database without an alias.
  SQL> -- SQL asks if you want to override the default:
  SQL> CREATE DATABASE FILENAME test;
  This alias has already been declared.
  Would you like to override this declaration (No)? NO
  %SQL-F-DEFDBDEC, A database has already been declared with the default
  alias
  ```

- In embedded SQL or in the SQL module language, specifying an already-declared alias in the CREATE DATABASE statement generates an error when you precompile the program or compile the module.
- In dynamic SQL, specifying an already-declared alias overrides the earlier declaration.

For more information about default databases, see Section 2.2.2.

**root-file-params-1**
**root-file-params-2**
**root-file-params-3**
Parameters that control the characteristics of the database root file or characteristics stored in the database root file that apply to the entire database. You can specify these parameters for either single-file or multifile databases.

Some database root file parameters specified in the CREATE DATABASE statement cannot be changed with the ALTER DATABASE statement. To change these database root file parameters, you must use the EXPORT and IMPORT statements. See the EXPORT Statement and the IMPORT Statement for information on exporting and importing your database.

**FILENAME file-spec**
The file specification associated with the database.

You can omit the FILENAME clause if you specify the ALIAS clause. If you omit the FILENAME clause, the file specification uses the following defaults:

- Device: the current device for the process (on OpenVMS only)
- Directory: the current directory for the process
- File name: the alias, if any was specified; otherwise omitting the FILENAME clause generates an error

Use either a full file specification or a partial file specification.

OpenVMS VAX OpenVMS Alpha You can use a logical name for all or part of a file specification. ♦

If you use a simple file name, SQL creates the database in the current default directory. Because the CREATE DATABASE statement may create more than one file with different file extensions, do not specify a file extension with the file specification.

The number and type of files created using the file specification in the FILENAME clause depend on whether you create a multifile or single-file database.

- In multifile CREATE DATABASE statements (any that include CREATE STORAGE AREA clauses), SQL uses the file specification to create up to three files:
  - A database root file with an .rdb file extension
  - A storage area file, with an .rda file extension, for the main storage area, RDB$SYSTEM, (unless the CREATE DATABASE statement contains a CREATE STORAGE AREA RDB$SYSTEM clause, which overrides this file specification)
  - A snapshot file, with an .snp file extension, for the main storage area, RDB$SYSTEM (unless the CREATE DATABASE statement contains a CREATE STORAGE AREA RDB$SYSTEM clause, which overrides this file specification)
- In single-file CREATE DATABASE statements (any that omit the CREATE STORAGE AREA clause), SQL uses the file specification to create two files:
  - A combined root and data file with an .rdb file extension
  - A snapshot file with an .snp file extension (unless overridden by a SNAPSHOT FILENAME clause in the storage area parameters)

If you create a single-file database, you cannot later create additional data and snapshot files with ALTER DATABASE . . . ADD STORAGE AREA statements. If you want to change a database from a single-file to a multifile database, you must use the EXPORT and IMPORT statements.

**PATHNAME path-name**

OpenVMS VAX OpenVMS Alpha

The repository path name for the repository directory where the database definition is stored.

Specify one of the following:

- A full repository path name, such as CDD$TOP.SQL.DEPT3
- A relative repository path name, such as DEPT3
- A logical name that refers to a full or relative repository path name

If you use a relative path name, CDD$DEFAULT must be defined as all the path name segments preceding the relative path name. For example, define CDD$DEFAULT as CDD$TOP.SQL, and then use the relative path name DEPT3.

```
SQL> SHOW DICTIONARY
The current data dictionary is  CDD$TOP.SQL
SQL> CREATE DATABASE ALIAS PERSONNEL PATHNAME DEPT3;
```

There is no default path name. If you do not specify a repository path name for the database, SQL does not store database definitions in the repository. Subsequent data definitions cannot use the repository. However, Oracle Rdb recommends that you do specify a repository path name when you create a database. For more information, see the Usage Notes in the DECLARE ALIAS Statement.

If you use the PATHNAME argument and your system does not have the repository, SQL ignores the argument.

When you use the PATHNAME argument, the repository associates the path name with the file specification exactly as given in the CREATE DATABASE statement. If that file specification is a file name, not a logical name, you cannot alter or delete the database by specifying the path name unless the database root file is in the current, default working directory.

The PATHNAME argument is available only on OpenVMS platforms. ♦

**literal-user-auth**

Specifies the user name and password for access to databases, particularly remote database.

This literal lets you explicitly provide user name and password information in the CREATE DATABASE statement.

**USER 'username'**
A character string literal that specifies the operating system user name that the database system uses for privilege checking. This clause also sets the value of the SYSTEM_USER value expression.

**USING 'password'**
A character string literal that specifies the user's password for the user name specified in the USER clause.

**DBKEY SCOPE IS ATTACH**
**DBKEY SCOPE IS TRANSACTION**
Controls when the database key of a deleted record can be used again by SQL. This setting is not strictly a database root file parameter, but a characteristic of the implicit database declaration executed by the CREATE DATABASE statement. Thus, the DBKEY SCOPE clause in a CREATE DATABASE statement takes effect only for the duration of the session of the user who entered the statement.

- The default DBKEY SCOPE IS TRANSACTION means that SQL can reuse the database key of a deleted table row (to refer to a newly inserted row) as soon as the transaction that deleted the original row completes with a COMMIT statement. (If the user who deleted the original row enters a ROLLBACK statement, then the database key for that row cannot be used again by SQL.)

  During the connection of the user who entered the CREATE DATABASE statement, the DBKEY SCOPE IS TRANSACTION clause specifies that a database key is guaranteed to refer to the same row *only* within a particular transaction.

  **Note**

  Oracle Rdb recommends using DBKEY SCOPE IS TRANSACTION to reclaim space on a database page faster than if you use DBKEY SCOPE IS ATTACH.

- The DBKEY SCOPE IS ATTACH clause means that SQL cannot use the database key again (to refer to a newly inserted row) until the user who deleted the original row detaches from the database (by declaring another database with the same alias or by using the DISCONNECT statement).

During the connection of the user who entered the CREATE DATABASE statement, the DBKEY SCOPE IS ATTACH clause specifies that a database key is guaranteed to refer to the same row until the user detaches from the database.

With the DBKEY SCOPE IS ATTACH clause, a user or program can complete one or several transactions and, while still attached to the database, use database keys (obtained through INSERT, DECLARE CURSOR, FETCH, or singleton SELECT statements) to directly access table rows with less locking and greater speed.

Remember that specifying the DBKEY SCOPE IS clause does not set a default database key scope characteristic for the database, but affects the database only for the duration of the session that created the database.

For more information, see Section 2.6.5.

**ROWID SCOPE IS ATTACH**
**ROWID SCOPE IS TRANSACTION**

The ROWID keyword is a synonym for the DBKEY keyword. See the DBKEY SCOPE IS argument earlier in this Arguments list for more information.

**MULTISCHEMA IS ON**
**MULTISCHEMA IS OFF**

Specifies the multischema attribute for the database. You must specify the multischema attribute for your database to create multiple schemas and store them in catalogs. Each time you attach to a database created with the multischema attribute, you can specify whether you want multischema naming enabled or disabled for subsequent statements. For more information on multischema naming, see Section 2.2.4.

If you prefer to access a database created with the multischema attribute as though it were single-schema database, you can turn off multischema naming using the MULTISCHEMA IS OFF clause in the ATTACH or DECLARE ALIAS statement.

If you have turned off the multischema attribute, you can enable it again using the MULTISCHEMA IS ON clause in the ATTACH or DECLARE ALIAS statement. You can use multischema naming only when you are attached to a database that was created with the multischema attribute. For more information, see the ATTACH Statement.

Multischema naming is disabled by default.

**OPEN IS MANUAL**
**OPEN IS AUTOMATIC**
Specifies whether or not the database must be explicitly opened before users can attach to it. The default, OPEN IS AUTOMATIC, means that any user can open a previously unopened or a closed database by attaching to it and executing a statement. The OPEN IS MANUAL option means that a privileged user must issue an explicit OPEN statement through Oracle RMU, the Oracle Rdb management utility, before other users can attach to the database.

The OPEN IS MANUAL option limits access to databases. You must have the DBADM privilege to attach to the database.

You receive an error message if you specify both OPEN IS AUTOMATIC and OPEN IS MANUAL options.

You can modify the OPEN IS option through the ALTER DATABASE statement.

**WAIT n MINUTES FOR CLOSE**
Specifies the amount of time that Oracle Rdb waits before automatically closing a database. If anyone attaches during that wait time, the database is not closed.

The default value for n is zero (0) if the WAIT clause is not specified. The value for n can range from zero (0) to 35,791,394. However, Oracle Rdb does not recommend using large values.

**PRESTARTED TRANSACTIONS ARE ON**
**PRESTARTED TRANSACTIONS ARE OFF**
Specifies whether Oracle Rdb enables or disables prestarted transactions.

Use the PRESTARTED TRANSACTIONS ARE OFF clause only if your application uses a server process that is attached to the database for long periods of time and causes the snapshot file to grow excessively. If you use the PRESTARTED TRANSACTIONS ARE OFF clause, Oracle Rdb uses additional I/O because each SET TRANSACTION statement must reserve a transaction sequence number (TSN).

For most applications, Oracle Rdb recommends that you enable prestarted transactions. The default is PRESTARTED TRANSACTIONS ARE ON. If you use the PRESTARTED TRANSACTIONS ARE ON clause or do not specify the PRESTARTED TRANSACTIONS clause, the COMMIT or ROLLBACK statement for the previous read/write transaction automatically reserves the TSN for the next transaction and reduces I/O.

The PRESTARTED TRANSACTIONS clause refers only to the database attach that is performed as part of the CREATE DATABASE statement. The clause does not set permanent database attributes.

You can define the RDMS$BIND_PRESTART_TXN logical name or the RDB_BIND_PRESTART_TXN configuration parameter to define the default setting for prestarted transactions outside of an application. The PRESTARTED TRANSACTION clause overrides this logical name or configuration parameter. For more information, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**RESTRICTED ACCESS**
**NO RESTRICTED ACCESS**
Restricts access to the database. This allows you to access the database but locks out all other users until you disconnect from the database. Setting restricted access to the database requires DBADM privileges.

The default is NO RESTRICTED ACCESS.

**COLLATING SEQUENCE sequence-name**
Specifies a default collating sequence to be used for all CHAR and VARCHAR columns in the database. SQL uses the default collating sequence if you do not specify a collating sequence in subsequent CREATE DOMAIN statements.

Sequence-name is a name of your choosing; you must use this name in any COLLATING SEQUENCE clauses that refer to this collating sequence for operations on this database.

**COMMENT IS 'string'**
Adds a comment about the collating sequence. SQL displays the text when it executes a SHOW COLLATING SEQUENCE statement in interactive SQL. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

**ncs-name**
The OpenVMS National Character Set (NCS) utility provides a set of predefined collating sequences and also lets you define collating sequences of your own. In the default NCS library, SYS$LIBRARY:NCS$LIBRARY, ncs-name is the name of a collating sequence or ncs-name is the name of the collating sequence in the NCS library specified by the library-name argument. (In most cases, it is simplest to make the collating sequence name the same as the ncs-name, for example, CREATE DATABASE . . . COLLATING SEQUENCE IS SPANISH SPANISH.) The COLLATING SEQUENCE clause accepts both predefined and user-defined NCS collating sequences.

If you omit the COLLATING SEQUENCE clause in the CREATE DATABASE statement at database definition time, the default sequence is the DEC Multinational Character Set (MCS).

**FROM library-name**
Specifies the name of an NCS library other than the default library. The default NCS library is SYS$LIBRARY:NCS$LIBRARY.

**NUMBER OF USERS number-users**
Specifies the maximum number of users allowed to access the database at one time. The default is 50 users. After the maximum is reached, the next user who tries to invoke the database receives an error message and must wait. The maximum number of users you can specify is 16368, and the minimum is 1 user.

Note that “number of users” is defined as the number of active attachments to the database. Thus, if a single process runs one program but that program performs 12 attach operations, the process is responsible for 12 active users as defined by this argument.

For information on how the NUMBER OF USERS parameter affects the NUMBER OF NODES parameter, see the Usage Notes.

**NUMBER OF BUFFERS number-buffers**
Specifies the number of buffers SQL allocates for each attach to this database. This number is displayed as the "default database buffer count" in the output from the RMU Dump command. The default buffer count applies to local and global buffers.

Specify an unsigned integer greater than or equal to 2 and less than or equal to 32,767. The default is 20 buffers.

**NUMBER OF CLUSTER NODES number-nodes**
Sets the upper limit on the maximum number of VMScluster nodes from which users can access the shared database. The default is 16 nodes. The range is 1 to 96 nodes. The maximum limit is the current VMScluster node limit set by your system administrator.

The NUMBER OF VAXCLUSTER NODES clause is retained for backward compatibility.

**NUMBER OF RECOVERY BUFFERS number-buffers**
Specifies the number of buffers allocated to the automatic recovery process that Oracle Rdb initiates after a system or process failure. This recovery process uses the recovery-unit journal (.ruj) file.

Specify an unsigned integer greater than or equal to 2 and less than or equal to 32,767. The default value for the NUMBER OF RECOVERY BUFFERS parameter is 40 buffers. If you have a large, multifile database and you are working on a system with a large amount of memory, specify a large number of buffers. This result is faster recovery time. However, make sure your buffer pool does not exceed the amount of memory you can allocate for the pool. if the number of buffers is too large for the amount of memory on your system, the system may be forced to perform virtual paging of the buffer pool. This can slow performance time because the operating system must perform the virtual paging of the buffer pool in addition to reading database pages. You may want to experiment to determine the optimal number of buffers for your database.

Use the NUMBER OF RECOVERY BUFFERS option to increase the number of buffers allocated to the recovery process.

```
SQL> CREATE DATABASE FILENAME personnel
cont>  NUMBER OF RECOVERY BUFFERS 150;
```

This option is used only if the NUMBER OF RECOVERY BUFFERS value is larger than the NUMBER OF BUFFERS value. For more information, see the *Oracle Rdb7 Guide to Database Maintenance.*

**BUFFER SIZE IS buffer-blocks BLOCKS**
Specifies the number of blocks SQL allocates per buffer. You need to specify an unsigned integer greater than zero. The default buffer size is 3 times the PAGE SIZE value (6 blocks for the default PAGE SIZE of 2).

The buffer size is a global parameter and the number of blocks per page (or buffer) is constrained to less than 64 blocks per page. The page size can vary by storage area for multifile databases, and the page size should be determined by the sizes of the records that will be stored in each storage area.

When choosing the number of blocks per buffer, choose a number so that a round number of pages fits in the buffer. In other words, the buffer size is wholly divisible by all page sizes for all storage areas in your multifile database. For example, if you have three storage areas with page sizes of 2, 3, and 4 blocks each respectively, choosing a buffer size of 12 blocks ensures optimal buffer utilization. In contrast, choosing a buffer size of 8 wastes 2 blocks per buffer for the storage area with a page size of 3 pages. Oracle Rdb reads as many pages as fit into the buffer; in this instance it reads two 3-block pages into the buffer, leaving 2 wasted blocks.

**GLOBAL BUFFERS ARE ENABLED**
**GLOBAL BUFFERS ARE DISABLED**
Specifies that Oracle Rdb maintains one global buffer pool per VMScluster node for each database. By default, Oracle Rdb maintains a local buffer pool

for each process. For more than one process to use the same page, each must read it from disk into its local buffer pool. A page in the global buffer pool may be read by more than one process at the same time, although only one process reads the page from the disk into the global buffer pool. Global buffering provides improved performance because I/O is reduced and memory is better utilized.

**Note**

In database parameter syntax, an attach to the database designates a **user**, and not necessarily the person who uses the database.

### NUMBER IS number-glo-buffers

Specifies the default number of global buffers to be used on one node when global buffers are enabled. This number appears as "global buffer count" in RMU Dump command output. Base this value on the database users' needs and the number of attachments. The default is the maximum number of attachments multiplied by 5.

**Note**

Do not confuse the NUMBER IS parameter with the NUMBER OF BUFFERS IS parameter. The NUMBER OF BUFFERS IS parameter determines the default number of buffers Oracle Rdb allocates to each user process that attaches to the database. The NUMBER OF BUFFERS IS parameter applies to, and has the same meaning for, both local and global buffering. The NUMBER IS parameter has meaning only within the context of global buffering.

You can override the default number of user-allocated buffers by defining a value for the logical name RDM$BIND_BUFFERS or configuration parameter RDB_BIND_BUFFERS. For more information, see the *Oracle Rdb7 Guide to Database Performance and Tuning.*

Although you can change the NUMBER IS parameter on line, the change does not take effect until the next time the database is opened.

### USER LIMIT IS max-glo-buffers

Specifies the maximum number of global buffers each attach allocates. Because global buffer pools are shared by all attachments, you must define an upper limit on how many global buffers a single attach can allocate. This limit prevents a user from defining the RDM$BIND_BUFFERS logical name or

RDB_BIND_BUFFERS configuration parameter to use all the buffers in the global buffer pool. (The behavior of RDM$BIND_BUFFERS or RDB_BIND_BUFFERS, which depends on whether you are using local or global buffers, is explained in the *Oracle Rdb7 Guide to Database Performance and Tuning*.)

The user limit cannot be greater than the total number of global buffers. The default is 5 buffers. The user limit appears as "maximum global buffer count per user" in RMU Dump command output.

Decide the maximum number of global buffers a process can allocate per attach by dividing the total number of global buffers set by the NUMBER IS clause by the total number of attachments for which you want to guarantee access to the database. For example, if the total number of global buffers is 200 and you want to guarantee at least 10 attachments access to the database, set the maximum number of global buffers per attach to 20.

In general, when you use global buffers, you should set the maximum global buffer count per user higher than the default database buffer count. For maximum performance on a VMScluster system, tune the two global buffer parameters on each node in the cluster using the RMU Open command with the Global_Buffers qualifier.

Although you can change the USER LIMIT IS parameter on line, the change does not take effect until the next time the database is opened.

The NUMBER IS and USER LIMIT IS parameters are the only two buffer parameters specific to global buffers. They are, therefore, in effect on a per node rather than a per process basis.

**PAGE TRANSFER VIA DISK**
**PAGE TRANSFER VIA MEMORY**
Specifies whether Oracle Rdb transfers (flushes) pages to disk or to memory.

When you specify PAGE TRANSFER VIA MEMORY, processes on a single node can share and update database pages in memory without transferring the pages to disk. It is not necessary for a process to write a modified page to disk before another process accesses the page.

The default is to DISK. If you specify PAGE TRANSFER VIA MEMORY, the database must have the following characteristics:

- The NUMBER OF CLUSTER NODES must equal one.
- GLOBAL BUFFERS must be enabled.
- After-image journaling must be enabled.
- FAST COMMIT must be enabled.

If the database does not have these characteristics, Oracle Rdb will perform page transfers via disk.

For more information about page transfers, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**SNAPSHOT IS ENABLED IMMEDIATE**
**SNAPSHOT IS ENABLED DEFERRED**
Specifies when read/write transactions write database changes they make to the snapshot file used by read-only transactions.

The default is ENABLED IMMEDIATE and causes read/write transactions to write copies of rows they modify to the snapshot file, regardless of whether or not a read-only transaction is active.

The ENABLED DEFERRED option lets read/write transactions avoid writing copies of rows they modify to the snapshot file (unless a read-only transaction is already active). Deferring snapshot writing in this manner improves the performance for the read/write transaction. However, read-only transactions that attempt to start after an active read/write transaction starts must wait for all active read/write users to complete their transactions.

**SNAPSHOT IS DISABLED**
Specifies that snapshot writing is disabled. Snapshot writing is enabled by default. If you specify the SNAPSHOT IS DISABLED option, you cannot specify either of the SNAPSHOT IS ENABLED options, and you cannot back up the database on line. You can, however, continue to set snapshot options in the event that you will enable snapshots in the future. SQL warns you of a possible conflict in the setting of snapshot options while snapshots are disabled, but SQL will execute the statement.

**DICTIONARY IS REQUIRED**
**DICTIONARY IS NOT REQUIRED**
OpenVMS VAX OpenVMS Alpha
Specifies whether or not definition statements issued for the database must also be stored in the repository. If you specify REQUIRED, any data definition statements issued after a DECLARE DATABASE statement that does not use the PATHNAME argument fails.

If you omit the PATHNAME clause from the database root file parameters in the CREATE DATABASE statement, SQL generates an error if you also specify DICTIONARY IS REQUIRED.

The default is DICTIONARY IS NOT REQUIRED.

The DICTIONARY clause is available only on OpenVMS platforms. ♦

**ADJUSTABLE LOCK GRANULARITY IS ENABLED**
**ADJUSTABLE LOCK GRANULARITY IS DISABLED**

Enables or disables whether or not the database system automatically maintains as few locks as possible on database resources. The default is ENABLED and results in fewer locks against the database. However, if contention for database resources is high, the automatic adjustment of locks can become a CPU drain. Such databases can trade more restrictive locking for less CPU usage by disabling adjustable lock granularity.

Always specify ADJUSTABLE LOCK GRANULARITY IS ENABLED if any table in your database contains more than 64,000 rows.

**COUNT IS n**

Specifies the number of levels on the page lock tree used to manage locks. For example, if you specify COUNT IS 3, the fanout factor is (10, 100, 1000). Oracle Rdb locks a range of 1000 pages and adjusts downward to 100 and then to 10 and then to 1 page when necessary.

If the COUNT IS clause is omitted, the default is 3. The value of n can range from 1 through 8.

**LOCK TIMEOUT INTERVAL IS number-seconds SECONDS**

Specifies the number of seconds for processes to wait during a lock conflict before timing out. The number of seconds can be between 1 and 65,000 seconds.

Specifying 0 is interpreted as no lock timeout interval being set. It is not interpreted as 0 seconds.

The lock timeout interval is database-wide; it is used as the default as well as the upper limit for determining the timeout interval. For example, if the database definer specified LOCK TIMEOUT INTERVAL IS 25 SECONDS in the CREATE DATABASE statement, and a user of that database specified SET TRANSACTION WAIT 30 or changed the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL or configuration parameter RDB_BIND_LOCK_TIMEOUT_INTERVAL to 30, SQL uses the interval of 25 seconds. For more information, see the SET TRANSACTION Statement and the *Oracle Rdb7 Guide to Distributed Transactions*.

**SEGMENTED STRING STORAGE AREA IS area-name**

Another name for LIST STORAGE AREA.

**LIST STORAGE AREA IS area-name**

Specifies the name of the storage area to be used for table columns defined through SQL with the LIST OF BYTE VARYING data type.

You can specify the LIST STORAGE AREA parameter for multifile databases only.

By default, columns with the LIST OF BYTE VARYING data type are stored in the RDB$SYSTEM storage area. If you specify a different storage area in this clause, the CREATE DATABASE statement must include a CREATE STORAGE AREA clause defining that area. For information about creating multiple list storage areas for a table, see the CREATE STORAGE AREA Clause.

**Note**

If you plan to store lists with segments of widely varying sizes, you should specify a MIXED page format area just for list storage. (Do not assign tables and indexes to the area.)

The database system looks for free space in an area when it stores each segment of a segmented string. If size varies significantly among the different segments of the lists that you plan to store, the interval and threshold values that the database system automatically sets for page format areas you specify as UNIFORM can make storing lists time-consuming. For a mixed page format area, you can customize interval and thresholds values to reduce the amount of time that the database system spends looking for free space when it stores different segments of the same segmented string.

---

The following example shows valid syntax for the LIST STORAGE AREA clause:

```
SQL> CREATE DATABASE FILENAME test
cont> LIST STORAGE AREA IS registry_area
cont>    CREATE STORAGE AREA RDB$SYSTEM FILENAME maintenance_area
cont>    CREATE STORAGE AREA registry_area FILENAME registry_area;
SQL> CREATE STORAGE MAP registry_map
cont> STORE LISTS IN registry_area;
```

**DEFAULT STORAGE AREA IS area-name**

Specifies a default storage area to which all user data and unmapped indexes are stored. The DEFAULT STORAGE AREA parameter separates user data from the system data, such as system tables. RDB$SYSTEM is the default area if you do not specify a default storage area.

In addition to user data, Oracle Rdb stores the following system tables in the default storage area:

- RDB$INTERRELATIONS

- RDB$MODULES
- RDB$ROUTINES
- RDB$PARAMETERS
- RDB$QUERY_OUTLINES
- Optional system tables, such as for multischema databases and the workload collection tables.

  For information on moving these system tables to other storage areas, see the *Oracle Rdb7 Guide to SQL Programming*.

The DEFAULT STORAGE AREA parameter must reference an existing storage area. You must create the storage area using the CREATE STORAGE AREA clause in the same CREATE DATABASE statement as the DEFAULT STORAGE AREA parameter.

**PROTECTION IS ANSI**
**PROTECTION IS ACLS**
Specifies whether the database root file will be invoked with ACL-style or ANSI/ISO-style privileges. If no protection clause is specified, the default is ACL-style privileges.

For ACL-style databases, the access privilege set is order-dependent. When a user tries to perform an operation on a database, SQL reads the associated access privilege set, called the access control list (ACL), from top to bottom, comparing the identifier of the user with each entry. As soon as SQL finds a match, it grants the rights listed in that entry and stops the search. All identifiers that do not match a previous entry “fall through” to the entry [ *,*] (equivalent to the SQL keyword PUBLIC). The default access for PUBLIC is NONE.

See the GRANT Statement and the REVOKE Statement for more information on ACL-style privileges.

For ANSI/ISO-style databases, the access privilege set is not order-dependent. The user matches the entry in the access privilege set; gets whatever privileges have been granted on the database, table, or column; and gets the privileges defined for PUBLIC. A user without an entry in the access privilege set gets only the privileges defined for PUBLIC. There is always an access privilege entry for PUBLIC, even if that entry has no access to the database, table, or column.

ANSI/ISO-style databases grant access to the creator when an object is created. Because only the creator is granted access to the newly created object, additional access must be granted explicitly.

See the GRANT Statement, ANSI/ISO-Style and the REVOKE Statement, ANSI/ISO-Style for more information on ANSI/ISO-style privileges.

You can change the PROTECTION IS parameter by using the IMPORT statement. See the IMPORT Statement for more information.

**RESERVE n CACHE SLOTS**
Specifies the number of row cache areas for which slots are reserved in the database. If your database is a single file database, you have only one cache slot and cannot reserve additional slots.

You can use the RESERVE CACHE SLOTS clause to reserve slots in the database root file for future use by the ADD CACHE clause of the ALTER DATABASE statement. Row cache areas can be added only if there are row cache area slots available. Slots become available after a DROP CACHE clause or a RESERVE CACHE SLOTS clause of the ALTER DATABASE statement.

The number of reserved slots for row cache areas cannot be decreased once the RESERVE clause is issued. If you reserve 10 slots and later reserve 5 slots, you have a total of 15 reserved slots for row cache areas.

If you do not specify the RESERVE CACHE SLOTS clause, the default number of row cache areas is one.

Reserving row cache slots is an offline operation (requiring exclusive database access). See the ALTER DATABASE Statement for more information about row cache areas.

**RESERVE n JOURNALS**
Specifies the number of journal files for which slots are reserved in the database. If your database is not a multifile database, you cannot reserve additional slots later using the ALTER DATABASE statement.

You must reserve slots before you can add journal files to the database.

See the ALTER DATABASE Statement for more information about adding journal files and enabling the journaling feature.

The following SQL statements create a multifile database and reserve 5 slots for future journal files.

```
SQL> CREATE DATABASE FILENAME test
cont>    RESERVE 5 JOURNALS
cont>    CREATE STORAGE AREA sa_one
cont>       ALLOCATION IS 10 PAGES;
```

### RESERVE n STORAGE AREAS

Specifies the number of storage areas for which slots are reserved in the database. The number of slots for storage areas must be a positive number greater than zero.

You can use the RESERVE STORAGE AREA clause to reserve slots in the database root file for future use by the ADD STORAGE AREA clause of the ALTER DATABASE statement. Storage areas can be added only if there are storage area slots available. Slots become available after a DROP STORAGE AREA clause or a RESERVE STORAGE AREA clause.

The number of reserved slots for storage areas cannot be decreased once the RESERVE clause is issued. If you reserve 5 slots and later reserve 10 slots, you have a total of 15 reserved slots for storage areas.

If you do not specify the RESERVE STORAGE AREA clause, the default number of storage areas is zero.

### SET TRANSACTION MODES

Enables only the modes specified, disabling all other previously defined modes. For example, if a database is to be used for read-only access and you want to disable all other transaction modes, use the following statement:

```
SQL> CREATE DATABASE FILENAME mf_personnel
cont>   SET TRANSACTION MODES (READ ONLY);
```

If not specified, the default transaction mode is ALL.

Specifying a negated transaction mode or specifying NONE disables all transaction usage. Disabling all transaction usage would be useful when, for example, you want to perform major restructuring of the physical database. Execute the ALTER DATABASE statement to re-enable transaction modes or use Oracle RMU, the Oracle Rdb management utility.

### ALTER TRANSACTION MODES

Enables the modes specified, leaving the previously defined or default modes enabled. For example, if the only transaction mode you want to disable are batch updates, use the following statement:

```
SQL> CREATE DATABASE FILENAME mf_personnel
cont>   ALTER TRANSACTION MODES (NO BATCH UPDATE);
```

If not specified, the default transaction mode is ALL.

**txn-modes**
Specifies the transaction modes for the database.

| Mode | Description |
|---|---|
| ALL | All modes are enabled. |
| NONE | No modes are enabled. |
| **Transaction Types** | |
| [NO]READ ONLY | Allows read-only transactions on the database. |
| [NO]READ WRITE | Allows read/write transactions on the database. |
| [NO] BATCH UPDATE | Allows batch-update transactions on the database. This mode executes without the overhead, or security, of a recovery-unit journal file. The batch-update transaction is intended for the initial loading of a database. Oracle Rdb recommends that this mode be disabled. |
| **Reserving Types** | |
| [NO] SHARED [READ \| WRITE] | Allows other users to work with the specified tables. |
| [NO] PROTECTED [READ \| WRITE] | Allows other users to read the specified tables. |
| [NO] EXCLUSIVE [READ \| WRITE] | Allows no access to other users to the specified tables. Access is exclusive to the user reserving the tables. |

For detailed information about the txn-modes, see the SET TRANSACTION Statement.

**CARDINALITY COLLECTION IS ENABLED**
**CARDINALITY COLLECTION IS DISABLED**
Specifies whether or not the optimizer records cardinality updates in the system table. When enabled, the optimizer collects cardinalities for the table and non-unique indexes as rows are inserted or deleted from tables. The update of the cardinalities is performed at commit time, if sufficient changes have accumulated, or at disconnect time.

In high update environments, it may be more convenient to disable cardinality updates. If you disable this feature, you should manually maintain the cardinalities using the RMU Analyze Cardinality command so the optimizer is given the most accurate values for estimation purposes.

Cardinality collection is enabled by default.

**CARRY OVER LOCKS ARE ENABLED**
**CARRY OVER LOCKS ARE DISABLED**
Enables or disables carry-over lock optimization. Carry-over locks are enabled by default.

While attached to the database, a process can have some active locks (locks attached to the database) and some carry-over locks (locks requested in earlier transactions that have not been demoted). If a transaction needs a lock it has currently marked as carry-over, it can reuse the lock by changing it to an active lock. The same lock can go from active to carry-over to active multiple times without paying the cost of lock request and demotion. This substantially reduces the number of lock requests if a process accesses the same areas repeatedly.

As part of the carry-over lock optimization, a NOWAIT transaction requests, acquires, and holds a NOWAIT lock. This signals other processes accessing the database that a NOWAIT transaction exists and causes Oracle Rdb to release all carry-over locks. If NOWAIT transactions are noticeably slow when executing, you can specify CARRY OVER LOCKS ARE DISABLED with the ALTER DATABASE or CREATE DATABASE statement.

This feature is available as an online database modification.

**LOCK PARTITIONING IS ENABLED**
**LOCK PARTITIONING IS DISABLED**
OpenVMS Alpha
Specifies whether more than one lock tree is used for the database or all lock trees for a database are mastered by one database resource tree.

When partitioned lock trees are enabled for a database, locks for storage areas are separated from the database resource tree and all locks for each storage area are independently mastered on the VMScluster node that has the highest traffic for that resource. OpenVMS determines the node that is using each resource the most and moves the resource hierarchy to that node.

You cannot enable lock partitioning for single-file databases. You should not enable lock partitioning for single-node systems, because all lock requests are local on single-node systems.

By default, lock partitioning is disabled.

This clause is available only on the OpenVMS Alpha platform. ♦

**METADATA CHANGES ARE ENABLED**
**METADATA CHANGES ARE DISABLED**

Specifies whether or not data definition changes are allowed to the database. This attribute becomes effective at the next database attach and affects all ALTER, CREATE, and DROP statements (except ALTER DATABASE, which is needed for database tuning) and the GRANT, REVOKE, and TRUNCATE TABLE statements. For example:

```
SQL> CREATE DATABASE FILENAME sample
cont> METADATA CHANGES ARE DISABLED;
SQL> CREATE TABLE t (a INTEGER);
SQL> DISCONNECT ALL;
SQL> ATTACH 'FILENAME sample';
SQL> CREATE TABLE s (b INTEGER);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-E-NOMETADATA, metadata operations are disabled
```

The METADATA CHANGES ARE DISABLED clause prevents data definition changes to the database. If you specify this clause in the CREATE DATABASE statement, system index compression is implicitly enabled.

The METADATA CHANGES ARE ENABLED clause allows data definition changes to the database by users granted the DBADMIN privilege.

METADATA CHANGES ARE ENABLED is the default.

**STATISTICS COLLECTION IS ENABLED**
**STATISTICS COLLECTION IS DISABLED**

Specifies whether the collection of statistics for the database is enabled or disabled. When you disable statistics for the database, statistics are not displayed for any of the processes attached to the database. Statistics are displayed using the RMU Show Statistics command.

The default is STATISTICS COLLECTION IS ENABLED. You can disable statistics using the ALTER DATABASE and IMPORT statements.

For more information on the RMU Show Statistics command, see the *Oracle RMU Reference Manual*.

You can enable statistics collection by defining the logical name RDM$BIND_STATS_ENABLED or the configuration parameter RDB_BIND_STATS_ENABLED. For more information about when to use statistics collection, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**SYSTEM INDEX COMPRESSION IS ENABLED**
**SYSTEM INDEX COMPRESSION IS DISABLED**
Specifies if you want Oracle Rdb to compress system indexes.

For system indexes, Oracle Rdb uses run-length compression, which compresses any sequences of two or more spaces from text data types or two or more binary zeros from nontext data types. Compressing system indexes results in reduced storage and improved I/O. Unless your applications frequently perform concurrent data definition, you should compress system indexes.

Once you create a database specifying the SYSTEM INDEX COMPRESSION clause, you only can change it using the EXPORT and IMPORT statements. You cannot alter the database to change the compression mode.

The default is SYSTEM INDEX COMPRESSION IS DISABLED.

**WORKLOAD COLLECTION IS ENABLED**
**WORKLOAD COLLECTION IS DISABLED**
Specifies whether or not the optimizer records workload information in the system table RDB$WORKLOAD. The WORKLOAD COLLECTION IS ENABLED clause creates this system table if it does not exist. If you later disable workload collection, the RDB$WORKLOAD system table is not deleted.

A workload profile is a description of the interesting table and column references used by queries in a database workload. When workload collection is enabled, the optimizer collects and records these references in the RDB$WORKLOAD system table. This work load is then processed by the RMU Analyze Statistics command which records useful statistics about the work load. These workload statistics are used by the optimizer at run time to deliver more accurate access strategies.

Workload collection is disabled by default.

**ASYNC BATCH WRITES ARE ENABLED**
**ASYNC BATCH WRITES ARE DISABLED**
Specifies whether asynchronous batch-writes are enabled or disabled.

Asynchronous batch-writes allow a process to write batches of modified data pages to disk asynchronously (the process does not stall while waiting for the batch-write operation to complete). Asynchronous batch-writes improve the performance of update applications without the loss of data integrity.

By default, batch-writes are enabled. For more information about when to use asynchronous batch-writes, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

You can enable asynchronous batch-writes by defining the logical name RDM$BIND_ABW_ENABLED or the configuration parameter RDB_BIND_ABW_ENABLED.

**CLEAN BUFFER COUNT IS buffer-count**
Specifies the number of buffers to be kept available for immediate reuse.

Oracle Rdb maintains the number of buffers at the end of a process' least recently used queue of buffers for replacement.

The default is five buffers. The minimum value is 1; the maximum value can be as large as the buffer pool size.

You can override the number of clean buffers by defining the logical name RDM$BIND_CLEAN_BUF_CNT or the configuration parameter RDB_BIND_CLEAN_BUF_CNT. For information about how to set the values, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**MAXIMUM BUFFER COUNT IS buffer-count**
Specifies the number of buffers a process will write asynchronously.

The default is one-fifth of the buffer pool, but not more than 10 buffers. The minimum value is 2 buffers; the maximum value can be as large as the buffer pool.

You can override the number of buffers to be written asynchronously by defining the logical name RDM$BIND_BATCH_MAX or the configuration parameter RDB_BIND_BATCH_MAX. For information about how to set the values, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**ASYNC PREFETCH IS ENABLED**
**ASYNC PREFETCH IS DISABLED**
Specifies whether or not Oracle Rdb reduces the amount of time that a process waits for pages to be read from disk by fetching pages before a process actually requests the pages.

Prefetch can significantly improve performance, but it may cause excessive resource usage if it is used inappropriately. Asynchronous prefetch is enabled by default. For more information about asynchronous prefetch, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

You can enable asynchronous prefetch by defining the logical name RDM$BIND_APF_ENABLED or the configuration parameter RDB_BIND_APF_ENABLED.

**DEPTH IS number-buffers BUFFERS**
Specifies the number of buffers to prefetch for a process.

The default is one-quarter of the buffer pool, but not more than eight buffers. You can override the number of buffers specified in the CREATE or ALTER DATABASE statements by using the logical name RDM$BIND_APF_DEPTH or the configuration parameter RDB_BIND_APF_DEPTH.

You can also specify this option with the DETECTED ASYNC PREFETCH clause.

**DETECTED ASYNC PREFETCH IS ENABLED**
**DETECTED ASYNC PREFETCH IS DISABLED**
Specifies whether or not Oracle Rdb reduces the amount of time that a process waits for pages to be read from disk.

By using heuristics, detected asynchronous prefetch determines if an I/O pattern is sequential in behavior even if sequential I/O is not actually executing at the time. For example, when a LIST OF BYTE VARYING column is fetched, the heuristics detect that the pages being fetched are sequential and, therefore, fetch ahead asynchronously to avoid wait times when the page is really needed.

Detected asynchronous prefetch is enabled by default.

**THRESHOLD IS number-pages PAGES**
Specifies the number of pages to prefetch for a process. The default is one-quarter of the buffer pool, but not more than eight pages.

If you specify the THRESHOLD option, you must have also specified the DETECTED ASYNC PREFETCH clause. You receive an error if you attempt to specify the THRESHOLD option with the ASYNC PREFETCH clause.

**ROW CACHE IS ENABLED**
**ROW CACHE IS DISABLED**
Specifies whether or not you want Oracle Rdb to enable the row caching feature.

When a database is created or is converted from a previous version of Oracle Rdb without specifying row cache support, the default is ROW CACHE IS DISABLED. Enabling row cache support does not affect database operations until a row cache area is created and assigned to one or more storage areas.

When the row caching feature is disabled, all previously created and assigned row cache areas remain in existence for future use when the row caching feature is enabled.

**LOCATION IS directory-spec**
Specifies the name of the backing store directory to which row cache information is written. The database system generates a file name (row-cache-name.rdc) automatically for each row cache area at checkpoint time. Specify a device name and directory name only, enclosed within single quotation marks. The file name is the row-cache-name specified when creating the row cache area. By default, the location is the directory of the database root file. These .rdc files are permanent database backing store files.

The LOCATION clause for a CREATE CACHE, ADD CACHE, or ALTER CACHE clause overrides this location, which is the default for the database.

**NO LOCATION**
Removes the location previously specified in a LOCATION IS clause for the row cache area. If you specify NO LOCATION, the row cache location becomes the directory of the database root file.

**INCREMENTAL BACKUP SCAN OPTIMIZATION**
**NO INCREMENTAL BACKUP SCAN OPTIMIZATION**
Specifies whether Oracle Rdb checks each area's SPAM pages or each database page to find changes during incremental backup.

If you specify INCREMENTAL BACKUP SCAN OPTIMIZATION, Oracle Rdb checks each area's SPAM pages and scans the SPAM interval of pages only if the SPAM transaction number (TSN) is higher than the root file backup TSN, which indicates that a page in the SPAM interval has been updated since the last full backup operation. Updates in the SPAM interval result in an extra I/O.

Specify INCREMENTAL BACKUP SCAN OPTIMIZATION if your database has large SPAM intervals or infrequently occurring updates, and you want to increase the speed of incremental backups.

If you specify NO INCREMENTAL BACKUP SCAN OPTIMIZATION, Oracle Rdb checks each page to find changes during incremental backup.

Specify the NO INCREMENTAL BACKUP SCAN OPTIMIZATION clause if your database has frequently occurring updates, uses bulk-load operations, or does not use incremental backups, or if you want to improve run-time performance.

The default is INCREMENTAL BACKUP SCAN OPTIMIZATION.

**MULTITHREAD AREA ADDITIONS**
Specifies whether Oracle Rdb creates all storage areas in parallel, creates a specified number in parallel, or creates areas serially.

This clause lets you determine the number of storage areas to be created in parallel, possibly saving time during the initial database creation. However, if you specify a large number of storage areas and many areas share the same device, multithreading may cause excessive disk head movement, which may result in the storage area creation taking longer than if the areas were created serially. In addition, if you specify a large number of storage areas, you may exceed process quotas, resulting in the database creation failing.

This setting is not saved as a permanent database attribute. It is used only during the execution of the CREATE DATABASE, ALTER DATABASE, or IMPORT statements.

If you do not specify the MULTITHREAD AREA ADDITIONS clause, the default is to create one storage area at a time. If you specify the MULTITHREAD AREA ADDITIONS clause, but do not specify an option, the default is all areas are created in parallel.

**ALL AREAS**
Specifies that all storage areas be created and initialized in parallel.

All storage areas are created asynchronously. If you are creating a large number of storage areas, you may exceed process quotas, resulting in the database creation failing.

**LIMIT TO n AREAS**
Specifies the number of storage areas to be created in parallel.

The number of areas should be smaller than the current process file open quota. The number of areas can range from between 1 and the number of storage areas being created.

**RECOVERY JOURNAL (LOCATION IS directory-spec)**
Specifies the location in which the recovery-unit journal (.ruj) file is written. Do not include node names, file names, or process-concealed logical names in the directory-spec. Single quotation marks are required around the directory-spec. This clause overrides the RDMS$RUJ logical name or the RDB_RUJ configuration parameter.

If this clause is omitted, the default directory location is either:

- The device:[RDM$RUJ] on OpenVMS or the location defined by the RDMS$RUJ logical name
- The database rootfile directory ( . . . /database.rdb/database.ruj) on Digital UNIX or the location defined by the RDB_RUJ configuration parameter.

See the *Oracle Rdb7 Guide to Database Maintenance* for more information on recovery-unit journal files.

OpenVMS VAX OpenVMS Alpha Following is an example using this clause on an OpenVMS system:

```
SQL> ALTER DATABASE FILENAME SAMPLE
cont> RECOVERY JOURNAL (LOCATION IS 'SQL_USER1:[DBDIR.RECOVER]');♦
```

Digital UNIX Following is an example using this clause on a Digital UNIX system:

```
SQL> ALTER DATABASE FILENAME sample
cont> RECOVERY JOURNAL (LOCATION IS '/tmp/dbdir');♦
```

**RECOVERY JOURNAL (NO LOCATION)**

Removes a location previously defined by a RECOVERY JOURNAL (LOCATION . . . ) clause or the location defined by the RDMS$RUJ logical name or the RDB_RUJ configuration parameter.

If you specify NO LOCATION, the recovery journal reverts to the default directory location device:[RDM$RUJ] on OpenVMS or to the database rootfile directory ( . . . /database.rdb/database.ruj) on Digital UNIX. See the *Oracle Rdb7 Guide to Database Maintenance* for more information on recovery-unit journal files.

**SHARED MEMORY IS SYSTEM**
**SHARED MEMORY IS PROCESS**

OpenVMS Alpha Determines whether database root global sections (including global buffers when enabled) are created in system space or process space. The default is PROCESS.

When you use global sections created in the process space, you and other users share physical memory and the OpenVMS operating system maps a row cache area to a private address space for each user. As a result, all users are limited by the free virtual address range and each use a percentage of memory in overhead. If many users are accessing the database, the overhead can be high. ♦

**storage-area-params**

Parameters that control the characteristics of database storage area files. You can specify most storage area parameters for either single-file or multifile databases, but the effect of the clauses differs.

- For single-file databases, the storage area parameters specify the characteristics for the single storage area in the database.

- For multifile databases, the storage area parameters specify a set of default values for any storage areas created by the CREATE DATABASE statement that do not specify their own values for the same parameters. The default values apply to the RDB$SYSTEM storage area, plus any others named in CREATE STORAGE AREA database elements.

  The CREATE STORAGE AREA clauses in a CREATE DATABASE statement can override these default values. The default values do not apply to any storage areas created later with the ALTER DATABASE statement.

**ALLOCATION IS number-pages**
The number of database pages allocated to the database initially. SQL automatically extends the allocation to handle the loading of data and subsequent expansion. Pages are allocated in groups of 3. An ALLOCATION of 25 pages would actually provide for 27 pages. The default is 402 pages. If you are loading a large database, a large allocation helps to prevent fragmented rows.

**CACHE USING row-cache-name**
Assigns the named row cache area as the default for all storage areas in the database. All rows stored in an area, whether they consist of table data, segmented string data, or special rows such as index nodes, are cached.

You must create the row cache area before terminating the CREATE DATABASE statement. For example:

```
SQL> CREATE DATABASE FILENAME test_db
cont> ROW CACHE IS ENABLED
cont> CACHE USING test1
cont> CREATE CACHE test1
cont>    CACHE SIZE IS 100 ROWS
cont> CREATE STORAGE AREA area1;
```

You can override the database default row cache area by either specifying the CACHE USING clause after the CREATE STORAGE AREA clause or by later altering the database and storage area to assign a new row cache area. Only one row cache area is allowed for each storage area.

If you do not specify the CACHE USING clause or the NO ROW CACHE clause, NO ROW CACHE is the default for the database.

**NO ROW CACHE**
Specifies that the database default is not to assign a row cache area to all storage areas in the database. You cannot specify the NO ROW CACHE clause if you specify the CACHE USING clause.

Alter the storage area and name a row cache area to override the database default. Only one row cache area is allowed for each storage area.

If you do not specify the CACHE USING clause or the NO ROW CACHE clause, NO ROW CACHE is the default for the database.

**EXTENT IS ENABLED**
**EXTENT IS DISABLED**
Enables or disables extents. Extents are enabled by default.

You can encounter performance problems when creating hashed indexes in storage areas with the mixed page format if the storage area was created specifying the wrong size for the area and if extents are enabled. By disabling extents, this problem can be diagnosed early and corrected to improve performance.

**EXTENT IS extent-pages**
**EXTENT IS (extension-options)**
Specifies the number of pages of each storage area file extent. For more information, see the SNAPSHOT EXTENT argument.

**MINIMUM OF min-pages PAGES**
Specifies the minimum number of pages of each extent. The default is 99 pages.

**MAXIMUM OF max-pages PAGES**
Specifies the maximum number of pages of each extent. The default is 9999 pages.

**PERCENT GROWTH IS growth**
Specifies the percent growth of each extent. The default is 20 percent growth.

**INTERVAL IS number-data-pages**
Specifies the number of data pages between space area management (SPAM) pages in the storage area file, and therefore the maximum number of data pages each space area management page will manage. The default, and also the minimum interval, is 216 data pages. The first page of each storage area is a space area management page. The interval you specify determines where subsequent space area management pages are to be inserted, provided there are enough data pages in the storage file to require more space area management pages.

You cannot specify the INTERVAL storage area parameter for single-file databases, and you cannot specify INTERVAL unless you also explicitly specify PAGE FORMAT IS MIXED.

Oracle Rdb calculates the maximum interval size based on the number of blocks per page and returns an error message if you exceed this value. For example, when the page size is 2 blocks, the maximum interval is 4008 pages. If you try to create a storage area with the interval set to 4009, Oracle Rdb returns the following error message:

```
%RDB-E-BAD_DPB_CONTENT, invalid database parameters in the database parameter
block (DPB)
-RDMS-F-SPIMAX, spam interval of 4009 is more than the Rdb maximum of 4008
-RDMS-F-AREA_NAME, area NEW
```

For more information about setting space area management parameters, see the *Oracle Rdb7 Guide to Database Maintenance*.

**LOCKING IS ROW LEVEL**
**LOCKING IS PAGE LEVEL**
Specifies page-level or row-level locking as the default for the database. This clause provides an alternative to requesting locks on records. You can override the database default lock level at the storage area level. The default is ROW LEVEL, which is compatible with previous versions of Oracle Rdb.

When many records are accessed in the same area and on the same page, the LOCKING IS PAGE LEVEL clause reduces the number of lock operations perfomed to process a transaction; however, this is at the expense of reduced concurrency. Transactions that benefit most with page-level locking are of short duration and also access several database records on the same page.

Use the LOCKING IS ROW LEVEL clause if transactions are long in duration and lock many rows.

The LOCKING IS PAGE LEVEL clause causes fewer blocking ASTs and provides better response time and utilization of system resources. However, there is a higher contention for pages and increased potential for deadlocks.

Page-level locking is *never* applied to RDB$SYSTEM, either implicitly or explicitly, because the lock protocol can stall metadata users.

You cannot specify page-level locking on single-file databases.

**PAGE FORMAT IS UNIFORM**
**PAGE FORMAT IS MIXED**
Specifies the on-disk structure for the storage area.

- The default is PAGE FORMAT IS UNIFORM and creates a storage area data file that is divided into **clumps**. Clump size, which is derived from buffer size, is 3 pages by default. A set of clumps forms a logical area that can contain rows from a single table only. For more information on uniform page formats, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

Uniform page format storage areas generally give the best performance if the tables in the storage area are subject to a wide range of queries.

- The PAGE FORMAT IS MIXED clause creates a storage area with a format that lets rows from more than one table reside on or near a particular page of the storage area data file. This is useful for storing related rows from different tables on the same page of the data file. For storage areas subject to repeated queries that retrieve those related rows, a mixed page format can greatly reduce I/O overhead if the mix of rows on the page is carefully controlled. However, mixed page format storage areas degrade performance if the mix of rows on the page is not suited for the queries made against the storage area.

**Note**

The main storage area created by the CREATE DATABASE statement, called RDB$SYSTEM, must have uniform pages. If you specify PAGE FORMAT IS MIXED as a default storage area parameter, SQL generates a warning message and overrides that default when it creates the RDB$SYSTEM storage area.

**PAGE SIZE IS page-blocks BLOCKS**
The size in blocks of each database page. Page size is allocated in 512-byte blocks. The default is 2 blocks (1024 bytes). If your largest row is larger than approximately 950 bytes, allocate more blocks per page to prevent fragmented rows. If you specify a page size larger than the buffer size, an error message is returned.

**CHECKSUM CALCULATION IS ENABLED**
**CHECKSUM CALCULATION IS DISABLED**
This option allows you to enable or disable calculations of page checksums when pages are read from or written to the storage area files.

The default is ENABLED.

**Note**

Oracle Rdb recommends that you leave checksum calculations enabled, which is the default.

With current technology, it is possible that errors may occur that the checksum calculation can detect but that may not be detected by either the hardware, firmware, or software. Unexpected application results and database corruption may occur if corrupt pages exist in memory or on disk but are not detected.

Oracle Rdb recommends performing checksum calculations, except in the following specific circumstances:

- Your application is stable and has run without errors on the current hardware and software configuration for an extended period of time.
- You have reached maximum CPU utilization in your current configuration. Actual CPU utilization by the checksum calculation depends primarily on the size of the database pages in your database. The larger the database page, the more noticeable the CPU usage by the checksum calculation may become.

**Note**

Oracle Rdb recommends that you carefully evaluate the trade-off between reducing CPU usage by the checksum calculation and the potential for loss of database integrity if checksum calculations are disabled.

Oracle Rdb allows you to disable and, subsequently, re-enable checksum calculation without error. However, once checksum calculations have been disabled, corrupt pages may not be detected even if checksum calculations are subsequently re-enabled.

**SNAPSHOT CHECKSUM CALCULATION IS ENABLED**
**SNAPSHOT CHECKSUM CALCULATION IS DISABLED**
Allows you to enable or disable calculations of page checksums when pages are read from or written to the snapshot files.

The default is ENABLED.

**Note**

Oracle Rdb recommends that you leave snapshot checksum calculations enabled, which is the default.

With current technology, it is possible that errors may occur that the snapshot checksum calculation can detect but that may not be detected by either the hardware, firmware, or software. Unexpected application results and database corruption may occur if corrupt pages exist in memory or on disk but are not detected.

Oracle Rdb recommends performing snapshot checksum calculations, except in the following specific circumstances:

- Your application is stable and has run without errors on the current hardware and software configuration for an extended period of time.
- You have reached maximum CPU utilization in your current configuration. Actual CPU utilization by the snapshot checksum calculation depends primarily on the size of the database pages in your database. The larger the database page, the more noticeable the CPU usage by the snapshot checksum calculation may become.

**Note**

Oracle Rdb recommends that you carefully evaluate the trade-off between reducing CPU usage by the snapshot checksum calculation and the potential for loss of database integrity if snapshot checksum calculations are disabled.

Oracle Rdb allows you to disable and, subsequently, re-enable snapshot checksum calculation without error. However, once snapshot checksum calculations have been disabled, corrupt pages may not be detected even if snapshot checksum calculations are subsequently re-enabled.

**SNAPSHOT ALLOCATION IS snp-pages PAGES**
Specifies the number of pages allocated for the snapshot file. The default is 99 pages.

**SNAPSHOT EXTENT IS extent-pages PAGES**
**SNAPSHOT EXTENT IS (extension-options)**
Specifies the number of pages of each snapshot or storage area file extent. The default extent for storage area files is 99 pages.

Specify a number of pages for simple control over the extension. For greater control, and particularly for multivolume databases, use the MINIMUM, MAXIMUM, and PERCENT GROWTH extension options instead.

If you use the MINIMUM, MAXIMUM, and PERCENT GROWTH parameters, you must enclose them in parentheses.

**SNAPSHOT FILENAME file-spec**
Provides a separate file specification for the storage area snapshot file. The SNAPSHOT FILENAME argument can only be used with a multifile database.

In a multifile database, the file specification is used for the RDB$SYSTEM storage area snapshot file, unless the CREATE DATABASE statement contains a CREATE STORAGE AREA RDB$SYSTEM clause that contains its own SNAPSHOT FILENAME clause.

Do not specify a file extension other than .snp to the snapshot file specification. Oracle Rdb will assign the extension .snp to the file specification, even if you specify an alternate extension.

If you omit the SNAPSHOT FILENAME argument, the .snp file gets the same device (on OpenVMS only), directory, and file name as the database root file.

**THRESHOLDS ARE (val1 [,val2 [,val3] ] )**
Specifies one, two, or three threshold values. The threshold values represent a fullness percentage on a data page and establish four possible ranges of guaranteed free space on the data pages. When a data page reaches the percentage defined by a given threshold value, the space area management (SPAM) entry for the data page is updated to reflect the new fullness percentage and its remaining free space.

The default thresholds are 70, 85, and 95 percent. If you specify only one or two values, unspecified values default to 100 percent.

You cannot specify the THRESHOLDS storage area parameter for single-file databases, and you cannot specify THRESHOLDS unless you also explicitly specify PAGE FORMAT IS MIXED. To specify thresholds for uniform storage areas, use the CREATE STORAGE MAP statement.

For more information about setting space area management parameters, see the *Oracle Rdb7 Guide to Database Maintenance*.

**WRITE ONCE**
The WRITE ONCE option of the storage-area-params clause permits you to create a storage area that contains only a segmented string in a format that can be stored on a write-once, read-many (WORM) optical device.

Oracle Rdb permits the storing of many write-once list segments on one write-once page, resulting in better write-once space usage. This improves storage performance because the storage algorithm reduces I/O due to more compact storage.

The following restrictions apply to the WRITE ONCE option:

- You cannot write data other than segmented strings to a write-once storage area. SQL issues an error message if you try to create a storage map that stores data other than segmented strings in a write-once storage area.
- When you create a storage area on WORM media, you must specify that the snapshot area remains on a read/write device; do not give a snapshot file the WRITE ONCE attribute.
- If you specify the WRITE ONCE option when storing a segmented string, database keys are not compressed. For more information on database key compression, see the *Oracle Rdb7 Guide to Database Maintenance*.
- WORM storage areas do not use SPAM pages. However, to assist moving data back to non-WORM devices on which SPAM pages must be built again, space is allocated for them. Because SPAM pages are essential in uniform areas, write-once storage areas cannot be of uniform format and, therefore, are required to be of mixed format.
- You can use the PAGE SIZE IS clause of the CREATE DATABASE statement to set the default page size for a storage area. To optimize storage, always specify an even number of blocks per page for a write-once storage area.
- Oracle Rdb does not support magnetic media for storing write-once storage areas.
- After you move a storage area to or from WORM media, back up your database completely and start a new after-image journal file. For more information on backup and recovery procedures with write-once storage areas, see the *Oracle Rdb7 Guide to Database Maintenance*.

  Oracle Rdb permits the storing of many write-once list segments on one write-once page, resulting in better write-once space usage. This improves storage performance because the storage algorithm reduces I/O due to more compact storage.

**JOURNAL IS ENABLED**
**JOURNAL IS DISABLED**
Specifies whether or not WRITE ONCE areas are written to the .aij file.

Disabling the journaling attribute on WRITE ONCE areas is beneficial because after-image journaling on storage media can slow the loading of large images or exceed storage area availability.

However, if there is a failure of the storage media, there may be loss of space or, more important, loss of information. In the case of a magnetic disk failure, the database is restored from an earlier backup and the AIJ records are applied to the restored database. There is no loss of information in this case, but could be loss of space because list of byte varying data written before the failure is not referenced by the existing data rows, and these list column values take up space on the write-once media that cannot be reused.

In the case of a WORM device failure, there can be loss of information because the existing data rows reference list column data that is no longer available. For example, if 120 pages were allocated in the WRITE ONCE area, and 100 pages had data written to the them, and the last backup was done when the area had 50 pages of information, any data on pages 51 to 120 is lost if there is a failure of the WORM device. Pages 51 to 120 are inaccessible. The RMU Repair command can be used to repair rows that reference missing list column data. For more information, see the *Oracle Rdb7 Guide to Database Maintenance* and the *Oracle RMU Reference Manual*.

Remember, the write-once storage area must be mixed format.

The default is JOURNAL IS ENABLED.

**DEFAULT CHARACTER SET support-char-set**
Specifies the database default character set for this database. For a list of allowable character set names, see Section 2.1.

**NATIONAL CHARACTER SET support-char-set**
Specifies the database national character set when you create a database. For a list of allowable national character set names, see Section 2.1.

**IDENTIFIER CHARACTER SET names-char-set**
Specifies the identifier character set for user-supplied database object names, such as table names and column names. The character set must contain ASCII characters. See Table 2-3 for a list of allowable character sets.

**database-element**
Database elements are a CREATE STORAGE AREA clause, any of the CREATE statements (except CREATE DOMAIN . . . FROM path-name and CREATE TABLE . . . FROM path-name), or a GRANT statement.

**create-cache-clause**
See the CREATE CACHE Clause for more details.

**create-catalog-statement**
See the CREATE CATALOG Statement for details.

If you want to specify a CREATE CATALOG statement in a CREATE DATABASE statement, you must first specify a MULTISCHEMA IS ON clause in the same CREATE DATABASE statement.

The CREATE CATALOG statement is committed immediately and cannot be rolled back. Before you specify the CREATE CATALOG statement, the following conditions must be true:

- The database is enabled for multischema.
- No transactions are active.
- The catalog alias must be the same as the database alias.

For information about enabling the database for multischema, see Section 2.2.3.

**create-collating-sequence-statement**
See the CREATE COLLATING SEQUENCE Statement for details.

If you want to specify a collating sequence in a CREATE DOMAIN statement embedded in a CREATE DATABASE statement, you must first specify a CREATE COLLATING SEQUENCE statement in the same CREATE DATABASE statement.

**create-domain-statement**
See the CREATE DOMAIN Statement for details.

OpenVMS VAX OpenVMS Alpha You cannot use the FROM path-name clause when embedding a CREATE DOMAIN statement in a CREATE DATABASE statement. You can, however, issue a separate CREATE DOMAIN statement following the CREATE DATABASE statement. You can also describe the domain directly in the CREATE DATABASE statement. ♦

If you want to specify a collating sequence in your embedded CREATE DOMAIN statement, you must first specify a CREATE COLLATING SEQUENCE statement in the same CREATE DATABASE statement.

**create-function-statement**
A CREATE FUNCTION statement. See the Create Routine Statement for details.

**create-index-statement**
See the CREATE INDEX Statement for details.

**create-module-statement**
See the CREATE MODULE Statement for details.

**create-procedure-statement**
A CREATE PROCEDURE statement. See the Create Routine Statement for details.

**create-schema-statement**
See the CREATE SCHEMA Statement for details.

The schema you create must have the same alias as the catalog and database that contain the schema, or they must share the default alias.

**create-storage-area-clause**
See the CREATE STORAGE AREA Clause for more details.

**create-storage-map-statement**
See the CREATE STORAGE MAP Statement for details.

**create-table-statement**
See the CREATE TABLE Statement for details.

OpenVMS VAX OpenVMS Alpha

You cannot use the FROM path-name clause when embedding a CREATE TABLE statement in a CREATE DATABASE statement. You can, however, issue a separate CREATE TABLE statement following the CREATE DATABASE statement. You can also describe the table directly in the CREATE DATABASE statement. ♦

The CREATE TABLE statements in a CREATE DATABASE statement can refer to domains not yet created, provided that CREATE DOMAIN statements for the domains are in the same CREATE DATABASE statement.

**create-trigger-statement**
See the CREATE TRIGGER Statement for details.

**create-view-statement**
See the CREATE VIEW Statement for details.

**grant-statement**
See the GRANT Statement for details.

## Usage Notes

- Unlike other data definition statements, the CREATE DATABASE statement does not start a transaction.
- You cannot roll back a CREATE DATABASE statement.

- You cannot issue the CREATE DATABASE statement when a transaction is active. If possible, make CREATE DATABASE the first SQL statement in a program or in an interactive session.
- A **context structure** is the data structure that describes the distributed transaction context. You cannot pass a context structure for a distributed transaction to a CREATE DATABASE statement because you cannot execute it when a transaction is already started.
- Although you cannot issue a CREATE DATABASE statement while a transaction is active, SQL lets you issue a CREATE DATABASE statement after a transaction is declared.

  When you do this, SQL automatically extends the scope of the currently declared transaction to include the new database. SQL uses the alias in the CREATE DATABASE statement and declares default transaction options (read/write, wait) for that alias. SQL preserves the transaction options for databases that were already part of the currently declared transaction.

OpenVMS VAX ≡ OpenVMS Alpha ≡

- By using the RDBVMS$CREATE_DB logical name and the RDBVMS$CREATE_DB identifier, you can restrict the ability of users to create databases on your system. For more information on the RDBVMS$CREATE_DB logical name and identifier, see the chapter on defining database protection in the *Oracle Rdb7 Guide to Database Design and Definition*. ♦
- The CREATE DATABASE statement creates a default access control list (ACL) for the database that gives the creator all SQL privileges to the database and no SQL privileges to all other users.
- When you create a database in a directory owned by a resource identifier, the access control entry for the directory is applied to the database rootfile ACL, and then the RMU access control entry is added. This is to prevent database users from overriding OpenVMS file security. However, this can result in a database that you consider your own, but to which you have no RMU access privileges.

  For more details and a workaround on this issue, see the *Oracle RMU Reference Manual* and the *Oracle Rdb7 Guide to Database Maintenance*.
- A process that requests more global buffers than the maximum is granted the maximum number of global buffers. This can cause slower performance than expected without any indication that something is wrong.

- The relationship between the number of users and the number of nodes supported on a database can cause unexpected output when you dump the database root file. For example, when you specify 2032 users and 4 nodes in an SQL CREATE or ALTER DATABASE statement and then dump the database root file, Oracle Rdb displays the values 2032 users and 41 nodes.

  Oracle Rdb uses a data structure called a TSN block (TSNBLK) to understand the relationship between the number of users and the number of nodes. A TSN block keeps track of transaction activity on a node and transaction information for each user on a particular node. Each TSN block is owned by a particular node and can handle up to 50 users. For each group of 50 users, one TSNBLK is allocated per node to cover the maximum number of users and VMScluster nodes the database is expected to support, which is determined as either one TSNBLK per VMScluster node or one TSNBLK per 50 users, whichever is larger. The maximum number of TSN blocks is equal to the value of the current maximum number of nodes that are supported for a database (currently 96) for Oracle Rdb.

  For example, if the database administrator (DBA) specifies 2032 users and 4 nodes, Oracle Rdb calculates this as 2032 divided by 50 for a total of 41 TSNBLKs, which equates to 41 nodes. The algorithm compares the number of nodes specified with the number of nodes calculated and selects the larger value. In this example, 41 is the maximum calculated value (the calculated 41 is greater than the specified 4).

  If the DBA specifies 2032 users and 50 nodes, 50 is the maximum value for the number of nodes (the specified 50 is greater than the calculated 41) and 50 TSNBLKs are allocated, one for each node.

  However, if the DBA specifies 50 users and 10 nodes, the maximum value is 10 nodes (the specified 10 is greater than the calculated 1), so 10 TSNBLKs are allocated, one for each node.

- If you attempt to define a database with the following collating sequence, Oracle Rdb returns an arithmetic exception error:

  ```
  native_2_upper_lower = cs(
  sequence = (%X00,"#"," ","A","a","B","b","C","c","D","d","E",
  "e","8","F","f","5"-"4","G","g","H","h","I","i","J","j","K","k",
  "L","l","M","m","N","n","9","O","o","1","P","p","Q","q","R","r",
  "S","s","7"-"6","T","t","3"-"2","U","u","V","v","W","w","X","x",
  "Y","y","Z","z"),
  modifications = (%X01-%X1F=%X00,"!"-""""=%X00,"$"-"0"=%X00,":"-"@"=
  %X00,
  "{"-%XFF=%X00,""="A"));
  ```

The modifications portion of the collating sequence results in too many characters being converted to NULL. Oracle Rdb can only handle about 80 character conversions to NULL.

A workaround is to modify the MULTINATIONAL2 character set to sort in the desired order.

- You cannot specify a snapshot file name for a single-file database.

  The SNAPSHOT FILENAME clause specified outside the CREATE STORAGE AREA clause is used to provide a default for subsequent CREATE STORAGE AREA statements. Therefore, this clause does not allow you to create a separate snapshot file for a single-file database (a database without separate storage areas).

  When you create a single-file database, Oracle Rdb does not store the file specification of the snapshot file. Instead, it uses the file specification of the root file (.rdb) to determine the file specification of the snapshot file.

  If you want to place the snapshot file on a different device or in a different directory, create a multifile database.

  OpenVMS VAX OpenVMS Alpha However, you can work around the restriction on OpenVMS platforms by defining a search list for a concealed logical name. (However, do not use a nonconcealed rooted logical name. Database files defined with a nonconcealed rooted logical name can be backed up, but do not restore as expected.)

  To create a database with a snapshot file on a different device or in a different directory:

  1. Define a search list using a concealed logical name. Specify the location of the root file as the first item in the search list and the location of the snapshot file as the second item.
  2. Create the database using the logical name for the directory specification.
  3. Copy the snapshot file to the second device or directory.
  4. Delete the snapshot file from the original location.

  If you are doing this with an existing database, close the database using the RMU Close command before defining the search list, and open the database using the RMU Open command after deleting the original snapshot file. Otherwise, follow the preceding steps.

An important consideration when placing snapshot and database files on different devices is the process of backing up and restoring the database. Use the RMU Backup command to back up the database. You can then restore the files by executing the RMU Restore command. Copy the snapshot file to the device or directory where you want it to reside, and delete the snapshot file from the location to which it was restored. For more information, see the *Oracle RMU Reference Manual.* ♦

- The following database definition can cause unexpected I/O to the WORM device and also lead to reduced performance:

  ```
  SQL> CREATE DATABASE FILENAME w
  cont>    LIST STORAGE AREA IS a1
  cont>    CREATE STORAGE AREA a1
  cont>     FILENAME a1
  cont>     WRITE ONCE
  cont>     PAGE FORMAT IS MIXED;
  ```

  This definition requests Oracle Rdb use the WORM storage area as the default list (segmented string) area. That is, all system table lists are stored in the WORM storage area. Because the type of segmented strings written for the system metadata are often revised (for example, a COMMENT IS or an ALTER statement) and often accessed at run time, they are unsuited for storage on a WORM device.

  Oracle Rdb issues the following message when you attempt to use a WORM storage area as the default list storage area:

```
%SQL-F-ERRCRESCH, Error creating database filename w
-RDB-E-BAD_DPB_CONTENT, invalid database parameters in the database parameter block (DPB)
-RDMS-E-DEFLISTWORM, default list (segmented string) storage area can not be a WRITE ONCE area
```

- You must set a dialect prior to creating a database if you wish to have extended character set support and you are specifying the default, national, or identifier character sets. See the SET DIALECT Statement for more information on setting a dialect.
- The database default character set specifies the character set for columns with CHAR and VARCHAR data types. For more information on the database default character set, see Section 2.1.3.
- The national character set specifies the character set for columns with the NCHAR and NCHAR VARYING data types. For more information on the national character set, see Section 2.1.4.
- The identifier character set specifies the character set for object names such as cursor names and table names. For more information on the identifier character set, see Section 2.1.2.

- If the DEFAULT CHARACTER SET clause is omitted, Oracle Rdb assumes that the database default character set is the default character set of the session within which the CREATE DATABASE statement is invoked if the dialect was previously set to SQL92 or MIA. Otherwise, the database default character set is DEC_MCS if this clause is omitted.
- If the NATIONAL CHARACTER SET clause is omitted, Oracle Rdb assumes that the national character set is the national character set of the session within which the CREATE DATABASE statement is invoked if the dialect was previously set to SQL92 or MIA. Otherwise, the national character set is DEC_MCS if this clause is omitted.
- If the IDENTIFIER CHARACTER SET clause is omitted, Oracle Rdb assumes that the identifier character set is the identifier character set of the session within which the CREATE DATABASE statement is invoked if the dialect was previously set to SQL92 or MIA. Otherwise, the identifier character set is DEC_MCS if this clause is omitted.
- OpenVMS VAX OpenVMS Alpha If the database default character set is not DEC_MCS, the PATHNAME specifier cannot be used due to a current limitation of the CDD/Repository, where object names must only contain DEC_MCS characters. SQL flags this as an error. ♦
- The database default, national, and identifier character sets cannot be changed after creation of the database.
- Oracle Rdb is not supported on Distributed File Server (DFS) disks. Because DFS does not support shared write, you cannot create a database on a DFS mounted disk. Oracle Rdb requires shared access because both the monitor and user need to open the root file simultaneously.
- CREATE DATABASE statements in programs must precede (in the source file) all other data definition language (DDL) statements that refer to the database.
- You cannot specify the COMMENT ON statement in a CREATE DATABASE statement.

```
SQL> CREATE DATABASE FILENAME test
cont> CREATE TABLE TEST_TABLES (COL1 REAL)
cont> COMMENT ON TABLE TEST_TABLES IS 'This won't work';
COMMENT ON TABLE TEST_TABLES IS 'This won't work';
^
%SQL-W-LOOK_FOR_STT, Syntax error, looking for:
%SQL-W-LOOK_FOR_CON,            IN, EDITPROC, VALIDPROC, GRANT, CREATE,
%SQL-W-LOOK_FOR_CON,            ;,
%SQL-F-LOOK_FOR_FIN,    found COMMENT instead
```

- If your database has snapshots set to ENABLED DEFERRED, users may not be able to attach to the database once you issue one of the following statements:
  - CREATE, ALTER, or DROP TABLE
  - CREATE or DROP INDEX

  During a database attach, Oracle Rdb locks certain key metadata and reads it to construct the metadata information cache used to process requests against the database. When one of the previously listed statements executes a read/write transaction that updates the metadata, any subsequent database attach (equivalent to a read-only transaction) will stall until the read/write transaction is completed. Users attached to the database before the statement was issued can continue accessing the database.

  Use of deferred snapshots will cause conflict when using data definition language (DDL) statements in a production environment because snapshot copies of the system metadata cannot be written to the snapshot file.

  To avoid this problem, modify the database so that snapshots are set to ENABLED IMMEDIATE. You can use any of the following statements to set snapshots to ENABLED IMMEDIATE:
  - CREATE DATABASE
  - ALTER DATABASE
  - IMPORT
- If you specify the WRITE ONCE (JOURNAL IS DISABLED) clause, a database that is recovered to a time prior to all transactions being committed causes old list of byte varying data to be visible again. If the database is recovered using a backup copy, access to some list of byte varying columns return an exception to indicate that old data is present on the write-once media.
- The maximum length for each string literal in a comment for a collating sequence is 1024 characters.
- Because of some special characteristics of the Norwegian collating sequence, certain restrictions apply when creating a Norwegian collating sequence in a database. The name of a Norwegian collating sequence in the NCS library must begin with the character string NORWEGIAN.

The sequence customarily shipped with OpenVMS is named NORWEGIAN, which meets this restriction. You may wish to alter the Norwegian sequence slightly or change its name. Oracle Rdb recommends that any variation of the Norwegian collating sequence be given a name such as NORWEGIAN_1 or NORWEGIANA.

- CREATE CACHE does not assign the row cache area to a storage area. You must use the CACHE USING clause with the CREATE STORAGE AREA clause of the CREATE DATABASE statement or the CACHE USING clause with the ADD STORAGE AREA or ALTER STORAGE AREA clauses of the ALTER DATABASE statement.
- The product of the CACHE SIZE and the ROW LENGTH settings determines the amount of memory required for the row cache area (some additional overhead and rounding up to page boundaries is performed by the database system). The row cache area is shared by all processes attached to the database from any node.
- The row cache area is shared by all processes attached to the database on any node.
- The following are requirements when using the row caching feature:
  - Fast commit must be enabled
  - Number of cluster nodes must equal 1
- Oracle Rdb recommends that you specify the UNIFORM page format for improved performance when specifying a default storage area.
- You cannot delete a storage area that has been established as the database default storage area.
- Setting the transaction mode to READ ONLY when creating a database prevents you from being able to define any database objects.
- Setting the NO BATCH UPDATE or NO EXCLUSIVE transaction modes prevents various transaction types on IMPORT and can effectively prevent the import from succeeding.
- Oracle Rdb prevents user specification of the disabled transactions modes when the transaction parameter block (TPB) is processed.
- You cannot enable after-image journaling or add after-image journal files with the CREATE DATABASE statement. You must use the ALTER DATABASE statement to enable after-image journaling or add after-image journal files.

## Examples

Example 1: Creating a single-file database

This command file example creates a single-file database that contains one table, EMPLOYEES, made up of domains defined within the CREATE DATABASE statement. The EMPLOYEES table has the same definition as that in the sample personnel database.

For an example that creates a multifile version of the personnel database, see the CREATE STORAGE AREA Clause.

```
SQL> -- By omitting a FILENAME clause, the database root file
SQL> -- takes the file name from the alias:
SQL> CREATE DATABASE ALIAS personnel
cont> --
cont> -- This CREATE DATABASE statement takes default
cont> -- database root file and storage area parameter values.
cont> --
cont> -- Create domains.
cont> -- Note that database elements do not terminate with semicolons.
cont> --
cont> CREATE DOMAIN ID_DOM CHAR(5)
cont> --
cont> CREATE DOMAIN LAST_NAME_DOM CHAR(14)
cont> --
cont> CREATE DOMAIN FIRST_NAME_DOM CHAR(10)
cont> --
cont> CREATE DOMAIN MIDDLE_INITIAL_DOM CHAR(1)
cont> --
cont> CREATE DOMAIN ADDRESS_DATA_1_DOM CHAR(25)
cont> --
cont> CREATE DOMAIN ADDRESS_DATA_2_DOM CHAR(20)
cont> --
cont> CREATE DOMAIN CITY_DOM CHAR(20)
cont> --
cont> CREATE DOMAIN STATE_DOM CHAR(2)
cont> --
cont> CREATE DOMAIN POSTAL_CODE_DOM CHAR(5)
cont> --
cont> CREATE DOMAIN SEX_DOM CHAR(1)
cont> --
cont> CREATE DOMAIN DATE_DOM DATE
cont> --
cont> CREATE DOMAIN STATUS_CODE_DOM CHAR(1)
cont> --
cont> -- Create a table:
cont> --
cont> CREATE TABLE EMPLOYEES
cont>    (
cont>    EMPLOYEE_ID      ID_DOM
cont>      CONSTRAINT     EMP_EMPLOYEE_ID_NOT_NULL
cont>      NOT NULL
cont>     NOT DEFERRABLE,
cont>    LAST_NAME        LAST_NAME_DOM,
cont>    FIRST_NAME       FIRST_NAME_DOM,
cont>    MIDDLE_INITIAL   MIDDLE_INITIAL_DOM,
cont>    ADDRESS_DATA_1   ADDRESS_DATA_1_DOM,
cont>    ADDRESS_DATA_2   ADDRESS_DATA_2_DOM,
cont>    CITY             CITY_DOM,
cont>    STATE            STATE_DOM,
cont>    POSTAL_CODE      POSTAL_CODE_DOM,
cont>    SEX              SEX_DOM,
cont>      CONSTRAINT     EMP_SEX_VALUES
cont>      CHECK          (
cont>                     SEX IN ('M', 'F') OR SEX IS NULL
cont>                     )
cont>     NOT DEFERRABLE,
cont>    BIRTHDAY         DATE_DOM,
cont>    STATUS_CODE      STATUS_CODE_DOM,
cont>      CONSTRAINT     EMP_STATUS_CODE_VALUES
cont>      CHECK          (
cont>                     STATUS_CODE IN ('0', '1', '2')
cont>                     OR STATUS_CODE IS NULL
cont>                     )
cont>     NOT DEFERRABLE,
cont>    )
cont> --
cont> -- End CREATE DATABASE by specifying a semicolon:
cont> ;
```

Example 2: Creating a database not using the repository

The following example:

- Creates the database root file acct.rdb in the default working directory
- Creates the snapshot file acct.snp in the default working directory
- Does not store the database definition in the repository
- Enables writing to the snapshot file

- Sets the allocation of the snapshot file to 200 pages

```
SQL>    CREATE DATABASE ALIAS acct
cont>     FILENAME acct
cont>     SNAPSHOT IS ENABLED IMMEDIATE
cont>     SNAPSHOT ALLOCATION IS 200 PAGES;
```

Example 3: Creating a database with the snapshot file disabled

This statement creates a database root file and, to save disk space, disables snapshot writing and sets the initial allocation size to 1.

```
SQL>    CREATE DATABASE ALIAS PERS
cont>     FILENAME personnel
cont>     SNAPSHOT IS DISABLED
cont>     ALLOCATION IS 1;
```

Example 4: Creating a database with ANSI/ISO-style privileges

This statement creates a database in which all ANSI/ISO-style privileges are granted to the creator of the database, WARRING, and no privileges are granted to the identifier [*,*], the PUBLIC identifier.

```
SQL>    CREATE DATABASE ALIAS EXAMPLE
cont>     FILENAME ansi_test
cont>     PROTECTION IS ANSI;
SQL>
SQL> SHOW PROTECTION ON DATABASE EXAMPLE;
Protection on Alias EXAMPLE
[SQL,WARRING]:
  With Grant Option:        SELECT,INSERT,UPDATE,DELETE,SHOW,CREATE,ALTER,DROP,
                            DBCTRL,OPERATOR,DBADM,SECURITY,DISTRIBTRAN
  Without Grant Option:     NONE
[*,*]:
  With Grant Option:        NONE
  Without Grant Option:     NONE
```

Example 5: Creating a database with a German collating sequence

This statement creates a database named LITERATURE and specifies a collating sequence named GERMAN (based on the GERMAN collating sequence defined in the NCS library).

```
SQL> CREATE DATABASE FILENAME literature
cont> COLLATING SEQUENCE GERMAN GERMAN;
SQL> SHOW COLLATING SEQUENCE
User collating sequences in schema with filename LITERATURE
     GERMAN
```

Example 6: Creating a database with global buffers

This statement creates a database named parts.rdb.

```
SQL> CREATE DATABASE ALIAS PARTS FILENAME parts
cont> GLOBAL BUFFERS ARE ENABLED (NUMBER IS 110, USER LIMIT IS 17);
```

Example 7: Creating a database specifying the database default and national character sets

The following SQL statements create a database specifying the database default character set of DEC_KANJI and the national character set of KANJI. Use the SHOW DATABASE statement to see the database settings.

```
SQL> SET DIALECT 'SQL92';
SQL> CREATE DATABASE FILENAME mia_char_set
cont>    DEFAULT CHARACTER SET DEC_KANJI
cont>    NATIONAL CHARACTER SET KANJI
cont>    IDENTIFIER CHARACTER SET DEC_KANJI;
SQL> --
SQL> SHOW CHARACTER SET;
Default character set is DEC_MCS
National character set is DEC_MCS
Identifier character set is DEC_MCS
Literal character set is DEC_MCS

Alias RDB$DBHANDLE:
        Identifier character set is DEC_KANJI
        Default character set is DEC_KANJI
        National character set is KANJI
```

See the SHOW Statement for information on the SHOW CHARACTER SETS statement.

Example 8: This example demonstrates how to:

- Create a multifile database
- Reserve slots for journal files, storage areas, and row caches
- Restrict access to the database for the current session
- Enable system index compression, row caching, and workload collection
- Disable statistics and cardinality collection
- Specify a default storage area
- Specify ROW as the lock-level default for the database
- Delay closing the database
- Create and assign a row cache area to a storage area

- Specify the location of the recovery-unit journal file

```
SQL> CREATE DATABASE FILENAME sample
cont>   SNAPSHOT IS DISABLED
cont>   RESERVE 10 JOURNALS
cont>   RESERVE 10 STORAGE AREAS
cont>   RESERVE 5 CACHE SLOTS
cont>   SYSTEM INDEX COMPRESSION IS ENABLED
cont>   ROW CACHE IS ENABLED
cont>   WORKLOAD COLLECTION IS ENABLED
cont>   RESTRICTED ACCESS
cont>   STATISTICS COLLECTION IS DISABLED
cont>   CARDINALITY COLLECTION IS DISABLED
cont>   LOCKING IS ROW LEVEL
cont>   DEFAULT STORAGE AREA IS area1
cont>   OPEN IS AUTOMATIC (WAIT 5 MINUTES FOR CLOSE)
cont>   RECOVERY JOURNAL (LOCATION IS 'SQL_USER1:[DAY]')
cont> CREATE CACHE cache1
cont>   CACHE SIZE IS 1000 ROWS
cont>   ROW LENGTH IS 1000 BYTES
cont> CREATE STORAGE AREA area1
cont>   CACHE USING cache1;
SQL>
SQL> SHOW DATABASE *;
Default alias:
    Oracle Rdb database in file sample
        Multischema mode is disabled
        Number of users:              50
        Number of nodes:              16
        Buffer Size (blocks/buffer):   6
        Number of Buffers:            20
        Number of Recovery Buffers:   20
        Snapshots are Disabled
        Carry over locks are enabled
        Lock timeout interval is 0 seconds
        Adjustable lock granularity is enabled (count is 3)
        Global buffers are disabled (number is 250, user limit is 5,
                  page transfer via disk)
        Journal fast commit is disabled
                ( checkpoint interval is 0 blocks,
                  checkpoint timed every 0 seconds,
                  no commit to journal optimization,
                  transaction interval is 256 )
        AIJ File Allocation:          512
        AIJ File Extent:              512
        Statistics Collection is DISABLED
        Unused Storage Areas:         10
        Unused Journals:              10
        System Index Compression is ENABLED
        Restricted Access
        Journal is Disabled
        Backup Server:   Manual
        Log Server:      Manual
        Overwrite:       Disabled
        Notification:    Disabled
        Asynchronous Prefetch is Enabled (depth is 5)
        Asynchronous Batch Write is Enabled (clean buffers 5, max buffers 4)
        Lock Partitioning is DISABLED
        Incremental Backup Scan Optim uses SPAM pages
        Shutdown Time is 60 minutes
        Unused Cache Slots:         5
        Workload Collection is Enabled
        Cardinality Collection is Disabled
        Metadata Changes are Enabled
        Row Cache is Enabled (Sweep interval is 1 second,
         No Location)
        Detected Asynch Prefetch is Enabled (depth is 4, threshold is 4)
        Default Storage Area AREA1
        Mode is Open Automatic (Wait 5 minutes for close)
        RUJ File Location SQL_USER1:[DAY]
        Database Transaction Mode(s) Enabled:
            ALL
        Dictionary Not Required
        ACL based protections
Storage Areas in database with filename sample
     RDB$SYSTEM                     List storage area.
     AREA1                          Default storage area.
Journals in database with filename sample
     No Journals Found
Cache Objects in database with filename sample
     CACHE1
SQL> SHOW CACHE cache1;

     CACHE1
        Cache Size:           1000 rows
        Row Length:           1000 bytes
        Row Replacement:      Enabled
        Shared Memory:        Process
        Large Memory:         Disabled
        Window Count:         100
        Reserved Rows:        20
        Sweep Rows:           3000
        No Sweep Thresholds
        Allocation:           100 blocks
        Extent:               100 blocks
```

## CREATE DOMAIN Statement

Creates a domain definition.

A domain defines the set of values, character set, collating sequence, and formatting clause that a column in a table can have. The CREATE DOMAIN statement specifies the set of values by associating a data type with a domain name. The CREATE and ALTER TABLE statements can use the domain in column definitions.

There are two ways to specify a domain definition:

- With a domain name, data type, and any combination of the following optional clauses:
  - Default value
  - Stored name
  - Collating sequence
  - SQL and DATATRIEVE formatting clauses
- OpenVMS VAX OpenVMS Alpha With the FROM clause and a repository path name that refers to a field already defined in the repository ♦

When the CREATE DOMAIN statement executes, SQL adds the domain definition to the database.

OpenVMS VAX OpenVMS Alpha If you attached to the database with the PATHNAME specification, the domain definition is also added to the repository. ♦

You can refer to a domain instead of an SQL data type in the CREATE and ALTER TABLE statements, and in formal parameter declarations in SQL module procedures. You can specify the same domain in many table definitions and in SQL module parameter declarations. If the domain has to change, you need only change that one domain definition (using the ALTER DOMAIN statement) to change all the tables and SQL modules that refer to it. This ability makes it easier to keep applications consistent.

SQL lets you specify a character data type or national character data type when defining a domain. It also lets you specify whether the length of the domain is measured in characters or octets.

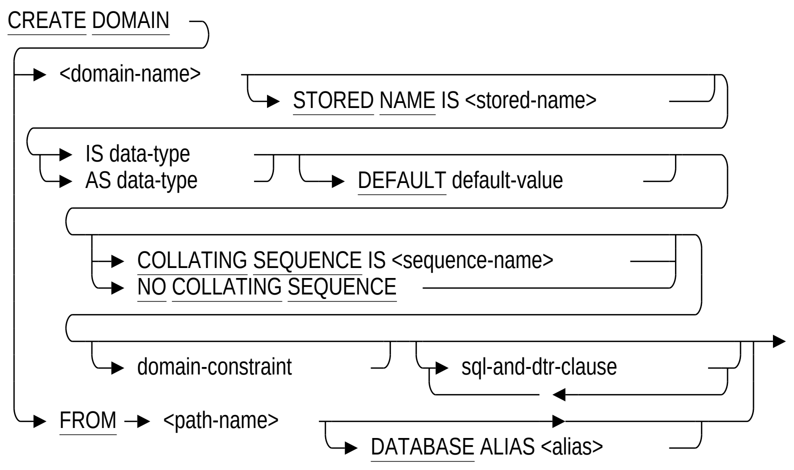
## CREATE DOMAIN Statement

### Environment

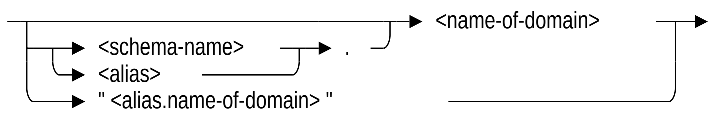
You can use the CREATE DOMAIN statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

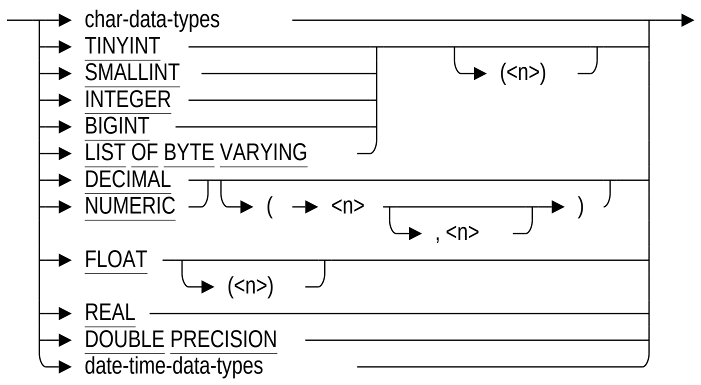
### Format



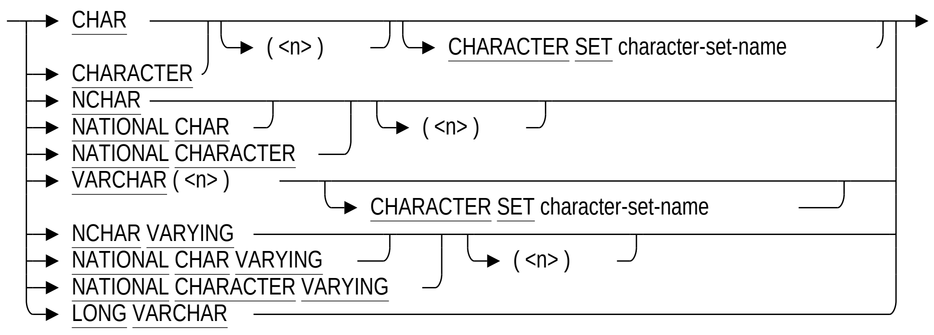
domain-name =



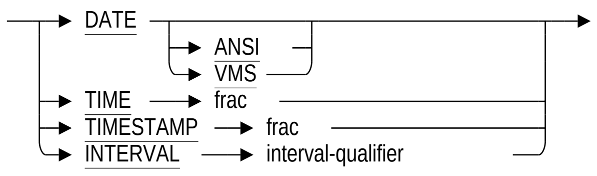
data-type =



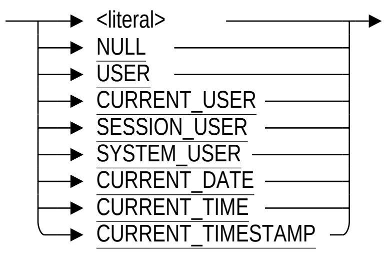
char-data-types =



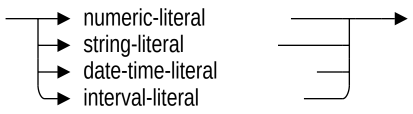
date-time-data-types =



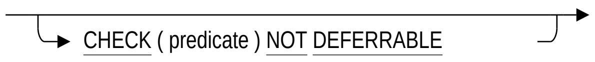
default-value =



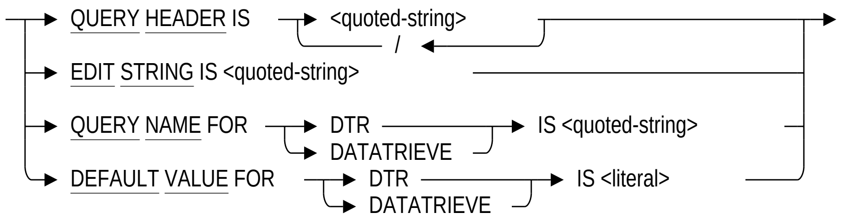
literal =



domain-constraint =



sql-and-dtr-clause =



## Arguments

**domain-name**
The name of a domain you want to create. The domain name must be unique among domain names in the schema. You can qualify it with an alias or (in multischema databases only) a schema name.

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access a domain created in a multischema database. The stored name lets you access multischema definitions using interfaces, such as Oracle RMU, the Oracle Rdb management utility, that do not recognize multiple schemas in one database. You cannot specify a stored name for a domain in a database that does not allow multiple schemas. For more information about stored names, see Section 2.2.4.

**IS data-type**
**AS data-type**
A valid SQL data type. See Section 2.3 for more information on data types.

**char-data-types**
A character type. See Section 2.3 for more information on data types.

**character-set-name**
A valid character set.

**date-time-data-types**
A data type that specifies a date, time, or interval. See Section 2.3.5 for more information about date-time data types.

**DEFAULT default-value**
A value to be stored in a column if the row that is inserted does not include a value for that column. You can use literals, the NULL keyword, the user name, the session user name, the system user name, the current date, the current time, or the current timestamp as default values. If you do not specify a default value, SQL assigns NULL as the default value.

**default-value**
Specifies the default value of a domain. The following table lists the valid values:

| Default Value | Description |
|---|---|
| literal | A value expression. Literal values can be numeric, character string, or date data types. |
| NULL | A null value. |
| USER | The current, active user name for a request. |

| Default Value | Description |
|---|---|
| CURRENT_USER | The current, active user name for a request. If a definer's rights request is executing, SQL returns the definer's user name. If not, SQL returns the session user name, if it exists. Otherwise, SQL returns the system user name. |
| SESSION_USER | The current, active session user name. If the session user name does not exist, SQL returns the system user name. |
| SYSTEM_USER | The user name of the process at the time of the database attach. |
| CURRENT_DATE | The DATE data type value containing year, month, and day for date "today". |
| CURRENT_TIME | The TIME data type value containing hours, minutes, and seconds for time "now". |
| CURRENT_ TIMESTAMP | The date and time currently defined in Oracle Rdb. |

OpenVMS VAX OpenVMS Alpha

**COLLATING SEQUENCE IS sequence-name**
Specifies a collating sequence for the named domain.

The OpenVMS National Character Set (NCS) utility provides a set of predefined collating sequences and also lets you define collating sequences of your own. The COLLATING SEQUENCE clause accepts both predefined and user-defined NCS collating sequences.

Before you use the COLLATING SEQUENCE clause in a CREATE DOMAIN statement, you must first specify the NCS collating sequence for SQL using the CREATE COLLATING SEQUENCE statement. The sequence-name argument in the COLLATING SEQUENCE clause must be the same as the sequence-name in the CREATE COLLATING SEQUENCE statement. ♦

OpenVMS VAX OpenVMS Alpha

**NO COLLATING SEQUENCE**
Specifies that this domain uses the standard default collating sequence, that is, ASCII. Use the NO COLLATING SEQUENCE clause to override the collating sequence defined for the database in the CREATE DATABASE or ALTER DATABASE statement. ♦

**domain-constraint**
Creates a constraint for the named domain.

Specify a domain constraint when you create a domain to limit which values can be stored in columns based on the domain. Domain constraints specify that columns based on the domain contain only certain data values or that data values can or cannot be null.

Use the CHECK clause to specify that a value must be within a specified range or that it matches a list of values. When you specify a CHECK clause for a domain constraint, you ensure that all values stored in columns based on the domain are checked consistently.

To refer to the values of all columns of a domain constraint, use the VALUE keyword. For example:

```
SQL> ALTER DOMAIN dom2
cont> ADD CHECK (VALUE IN ('F','M'))
cont> NOT DEFERRABLE;
```

For any dialect other than SQL92, you must specify that domain constraints are NOT DEFERRABLE.

When you create a domain constraint, SQL propagates the new constraint definition to all the columns that are based on the domain. If columns that are based on the domain contain data that does not conform to the constraint, SQL returns the following error:

```
%RDB-E-NOT_VALID, validation on field DATE_COL caused operation to fail
```

**sql-and-dtr-clause**
Optional SQL and DATATRIEVE formatting clause. See Section 2.5 for more information on formatting clauses.

You cannot use the clauses beginning with *NO* with the CREATE DOMAIN statement. They are valid only with the ALTER TABLE and ALTER DOMAIN statements.

OpenVMS VAX OpenVMS Alpha

**FROM path-name**
Specifies the repository path name of a repository field definition. SQL creates the domain using the definition from this field and gives the domain the name of the field definition.

Creating a domain based on a repository domain definition is useful when many applications share the same definition. Changes to the common definition can be automatically reflected in all applications that use it.

You can create a domain using the FROM path-name clause only if the field definition in the repository was originally created using the repository CDO utility. For instance, you cannot create a domain using the FROM path-name clause if the definition was created in the repository as part of an SQL session. Oracle Rdb requires that the field names referenced in the VALID IF expression of the CDO utility match the name of the global field being defined or changed.

**Note**

Changes by other users or applications to the field definition in the repository will affect the domain definition once the database is integrated to match the repository with an INTEGRATE DATABASE . . . ALTER FILES statement.

---

You can use the FROM path-name clause only if the database was attached specifying PATHNAME. You can specify either a full repository path name or a relative repository path name.

You cannot specify formatting clauses when you use the FROM path-name form of the CREATE DOMAIN statement.

You cannot use the FROM path-name clause when embedding a CREATE DOMAIN statement in a CREATE DATABASE statement.

The FROM path-name argument can be specified only on OpenVMS platforms. ♦

OpenVMS VAX OpenVMS Alpha

**DATABASE ALIAS alias**
In the FROM path-name clause, specifies the name for an attach to a particular database. SQL adds the domain definition to the database referred to by the alias.

If you do not specify an alias, SQL adds the domain definition to the default database. See Section 2.2.2 for more information on default databases and aliases.

The DATABASE ALIAS clause can be specified only on OpenVMS platforms. ♦

## Usage Notes

- You must execute the CREATE DOMAIN statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with characteristics specified in the most recent DECLARE TRANSACTION statement.

- You cannot execute the CREATE DOMAIN statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on the RDB$SYSTEM storage area.
- In general, you should use domains when creating tables because domains:
  - Ensure that similar columns in multiple tables comply to one standard. For example, if you define the columns using the domain ID_DOM, the data type for all these columns will be CHAR(5).
  - Let you change the data type or DATATRIEVE parameters for all columns defined using a domain, by changing the domain itself. For example, if you want to change the data type for the domain POSTAL_CODE_DOM from CHAR(5) to CHAR(10), you need alter only the data type for POSTAL_CODE_DOM. You do not have to alter the data type for the column POSTAL_CODE in the tables COLLEGES and EMPLOYEES.
  - Let you specify default values for all columns that were defined using a domain. For example, you can use a value such as NULL or Not Applicable that clearly demonstrates that no data was inserted into a column based on that domain. If a column usually contains a particular value, you can use that value as the default. For example, if most company employees live in the same state, you could make that state the default value for the STATE_DOM column.

    A default value specified for a column overrides a default value specified for the domain.

    Remember that the default value is not the same as the missing value that you can specify using the RDO interface. In contrast to default values, changing the missing value does change what is displayed by applications based on RDO for columns that have no data value stored and that have a missing value defined. See the *Oracle Rdb7 Guide to Database Design and Definition* for the difference between default value and missing value.
- The data type of a value specified in the DEFAULT VALUE clause must be the same data type as the column in which it is defined. If you forget to specify the data type, SQL issues an error message, as shown in the following example:

```
SQL> CREATE DOMAIN TIME_DOM IS TIME ( 2 ) DEFAULT '00:00:00.00' ;
%SQL-F-DEFVALINC, You specified a default value for TIME_DOM which is
inconsistent with its data type
SQL> CREATE DOMAIN TIME_DOM IS TIME ( 2 ) DEFAULT TIME '00:00:00.00' ;
```

- The result data type for the USER, CURRENT_USER, SESSION_USER, and SYSTEM_USER keywords is CHAR(31).
- You might not want to use domains when you create tables if:
  - Your application must be compatible with the current ANSI/ISO SQL standard. Domains are not standard in the ANSI/ISO SQL standard. They are expected to be included in the next version of the ANSI/ISO SQL standard.
  - You are creating intermediate result tables. It takes time to plan what the domains are in the database and to define them. Intermediate result tables might not warrant this effort.
- OpenVMS VAX OpenVMS Alpha The CREATE DOMAIN statement fails when both of the following are true:
  - The database to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The database was attached using the FILENAME argument.

  Under these circumstances, the statement fails with the following error when you issue it:

```
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-CDDISREQ, CDD required for metadata updates
                  is not being maintained
```

  ♦
- OpenVMS VAX OpenVMS Alpha It is possible when using the repository to define field structures that are not acceptable to Oracle Rdb.

  The repository is intended as a generic data repository that can hold data structures available to many layered products and languages.

  These data structures may not always be valid when applied to the relational data model used by Oracle Rdb.

  The following are some of the common incompatibilites between the data structures of the repository and Oracle Rdb.
  - %CDD-E-PRSMISSNG, attribute value is missing

    This error can occur when a field definition in the repository contains a FILLER clause.

  - %CDD-E-INVALID_RDB_DTY, datatype of field is not supported by Oracle Rdb

    This error can occur when a field definition in the repository contains an ARRAY clause.

  - %CDD-E-HAS_DIMENSION, Oracle Rdb fields cannot have dimension

    This error can occur when a field definition in the repository contains an OCCURS clause. ♦

- You can specify the national character data type by using the NCHAR, NATIONAL CHAR, NCHAR VARYING, or NATIONAL CHAR VARYING data types. The national character data type is defined by the database national character set when the database is created. See Section 2.3.1 for more information regarding national character data types.
- You can specify the length of the data type in characters or octets. By default, data types are specified in octets. By preceding the CREATE DOMAIN statement with the SET CHARACTER LENGTH 'CHARACTERS' or SET DIALECT 'MIA' or SET DIALECT 'SQL92' statement, you change the length to characters. For more information, see the SET CHARACTER LENGTH Statement and the SET DIALECT Statement.
- You can create a domain without specifying a character set that defines the domain with the database default character set.
- You can create a domain specifying a character set other than the database default or national character sets.
- When creating a domain constraint, the predicate cannot contain subqueries and cannot refer to another domain.
- You can only specify one constraint for each domain.
- Because of some special characteristics of the Norwegian collating sequence, certain restrictions apply when creating a Norwegian collating sequence in a database. The name of a Norwegian collating sequence in the NCS library must begin with the character string NORWEGIAN.

  Please note that the sequence customarily shipped with OpenVMS is named NORWEGIAN, which meets this restriction. You may wish to alter the Norwegian sequence slightly or change its name. Oracle recommends that any variation of the Norwegian collating sequence be given a name such as NORWEGIAN_1 or NORWEGIANA.

- The CHECK constraint syntax can reference the VALUE keyword or the domain name. For example:

```
SQL> -- The CHECK constraint can reference the VALUE keyword.
SQL> --
SQL> CREATE DOMAIN D1 INTEGER
cont> CHECK (VALUE > 10)
cont> NOT DEFERRABLE;
SQL> SHOW DOMAIN D1;
D1                              INTEGER
 Valid If:       (VALUE > 10)
SQL> ROLLBACK;
SQL> --
SQL> -- The CHECK constraint can reference the domain name.
SQL> --
SQL> CREATE DOMAIN D1 INTEGER
cont> CHECK (D1 > 10)
cont> NOT DEFERRABLE;
SQL> SHOW DOMAIN D1
D1                              INTEGER
 Valid If:       (D1 > 10)
```

## Examples

Example 1: Creating a domain for a standard EMPLOYEE_ID definition

The following example creates the domain ID_DOM, which will be a standard definition of columns for the employee ID:

```
SQL> CREATE DOMAIN ID_DOM CHAR(5);
SQL> COMMENT ON DOMAIN ID_DOM IS
cont> 'standard definition of employee id';
```

Example 2: Creating a domain for standard date

The following example creates the domain STANDARD_DATE_DOM, which includes the edit string DD-MMM-YYYY:

```
SQL> CREATE DOMAIN STANDARD_DATE_DOM DATE
cont> EDIT STRING IS 'DD-MMM-YYYY';
SQL> COMMENT ON DOMAIN STANDARD_DATE_DOM IS
cont> 'standard definition for complete dates';
```

Example 3: Creating domains with default values

The following example creates two domains: ADDRESS_DATA2_DOM and WORK_STATUS_DOM. The ADDRESS_DATA2_DOM domain has a default value of NULL; the WORK_STATUS_DOM domain has a default value of 1 to signify full-time work status.

```
SQL> CREATE DOMAIN ADDRESS_DATA2_DOM CHAR(20)
cont> DEFAULT NULL;
SQL> --
SQL> CREATE DOMAIN WORK_STATUS_DOM SMALLINT
cont> DEFAULT 1;
```

OpenVMS VAX OpenVMS Alpha

Example 4: Basing a domain on a repository field definition

The following example illustrates using the repository as a source for the definition in a CREATE DOMAIN statement:

```
$ DICTIONARY OPERATOR
Welcome to CDO V2.3
The CDD/Repository V5.3 User Interface
Type HELP for help
CDO> DEFINE FIELD domain_test DATATYPE IS SIGNED QUADWORD.
CDO> EXIT
$ SQL :== $SQL$
$ SQL
SQL> ATTACH 'PATHNAME CDD$TOP.SQL.RDB.TEST.DATE';
SQL> CREATE DOMAIN FROM DOMAIN_TEST;
SQL> SHOW DOMAIN
User domains in database with pathname
        SYS$COMMON:[CDDPLUS]SQL.RDB.TEST.DATE;1
DOMAIN_TEST                     QUADWORD
```

♦

Example 5: Creating a domain with a collating sequence

The following example creates a domain with the predefined NCS collating sequence SPANISH. Note that you must first execute the CREATE COLLATING SEQUENCE statement:

```
SQL> --
SQL> CREATE COLLATING SEQUENCE SPANISH SPANISH;
SQL> CREATE DOMAIN LAST_NAME_SPANISH CHAR(14)
cont> COLLATING SEQUENCE IS SPANISH;
SQL> --
SQL> SHOW DOMAIN LAST_NAME_SPANISH
LAST_NAME_SPANISH               CHAR(14)
 Collating sequence: SPANISH
```

Example 6: Creating a domain using the database default character set

For each of the following examples, assume the database was created specifying the database default character set as DEC_KANJI and the national character set as KANJI.

The following example creates the domain DEC_KANJI_DOM using the database default character set:

```
SQL> SHOW CHARACTER SET;
Default character set is DEC_KANJI
National character set is KANJI
Identifier character set is DEC_KANJI
Literal character set is DEC_KANJI

Alias RDB$DBHANDLE:
        Identifier character set is DEC_KANJI
        Default character set is DEC_KANJI
        National character set is KANJI
SQL> CREATE DOMAIN DEC_KANJI_DOM CHAR(8);
SQL> SHOW DOMAIN
User domains in database with filename MIA_CHAR_SET
DEC_KANJI_DOM                   CHAR(8)
```

Because the CREATE DOMAIN statement does not specify a character set, Oracle Rdb defines the domain using the database default character set. The database default character set does not display with the SHOW DOMAIN statement.

An equivalent statement to the previous CREATE DOMAIN statement is:

```
SQL> CREATE DOMAIN DEC_KANJI_DOM CHAR(8) CHARACTER SET DEC_KANJI;
```

Example 7: Creating a domain using the national character set

The following example creates the domain KANJI_DOM using the NCHAR data type to designate use of the national character set:

```
SQL> CREATE DOMAIN KANJI_DOM NCHAR(8);
SQL> SHOW DOMAIN
User domains in database with filename MIA_CHAR_SET
DEC_KANJI_DOM                   CHAR(8)
KANJI_DOM                       CHAR(8)
         KANJI 8 Characters,  16 Octets
```

When a character set other than the default is specified, the SHOW DOMAIN statement displays the character set associated with the domain.

Two statements equivalent to the previous CREATE DOMAIN statement are:

```
SQL> CREATE DOMAIN KANJI_DOM NATIONAL CHAR(8);
SQL> CREATE DOMAIN KANJI_DOM CHAR(8) CHARACTER SET KANJI;
```

Example 8: Creating a domain constraint

The following example creates a domain constraint:

```
SQL> -- The SET DIALECT 'SQL92' statement sets the default date format
SQL> -- to the ANSI/ISO SQL standard format.
SQL> --
SQL>  SET DIALECT 'SQL92';
SQL> --
SQL> -- The following domain ensures that any dates inserted into the database
SQL> -- are later than January 1, 1900:
SQL> --
SQL> CREATE DOMAIN TEST_DOM DATE
cont>       DEFAULT NULL
cont>       CHECK (VALUE > DATE'1900-01-01' OR
cont>              VALUE IS NULL)
cont>           NOT DEFERRABLE;
SQL>
SQL> -- The following example creates a table with one column based on the
SQL> -- domain TEST_DOM:
SQL> --
SQL> CREATE TABLE DOMAIN_TEST
cont>       (DATE_COL   TEST_DOM);
SQL> --
SQL> -- SQL returns an error if you attempt to insert data that does not
SQL> -- conform to the domain constraint:
SQL> --
SQL> INSERT INTO DOMAIN_TEST
cont>   VALUES (DATE'1899-01-01');
%RDB-E-NOT_VALID, validation on field DATE_COL caused operation to fail
```

## CREATE FUNCTION Statement

Creates an external function as a schema object in an Oracle Rdb database.

The CREATE FUNCTION statement is documented under the Create Routine Statement. For complete information on creating an external function definition, see the Create Routine Statement.

## CREATE INDEX Statement

Creates an index for a table. An index allows direct access to the rows in the table to avoid sequential searching.

You define an index by listing the columns in a table that make up the index. You can define more than one index for a table. The index can be made up of one column, or two or more columns. An index made up of two or more columns is called a **multisegmented index**.

Optional arguments to the CREATE INDEX statement let you specify:

- The type of index structure (hashed, sorted nonranked, or sorted ranked)
- The names of a storage area or storage areas that contain the index
- Physical characteristics of a sorted index structure, such as index node size and the initial fullness percentage of each node
- Compression characteristics, including compressed key suffixes for text indexes and integer column compression for word or longword numeric columns
- Compression of space characters from text data types and of binary zeros from nontext data types
- Duplicates handling for sorted, ranked indexes
- Thresholds for the logical storage areas that contain the index

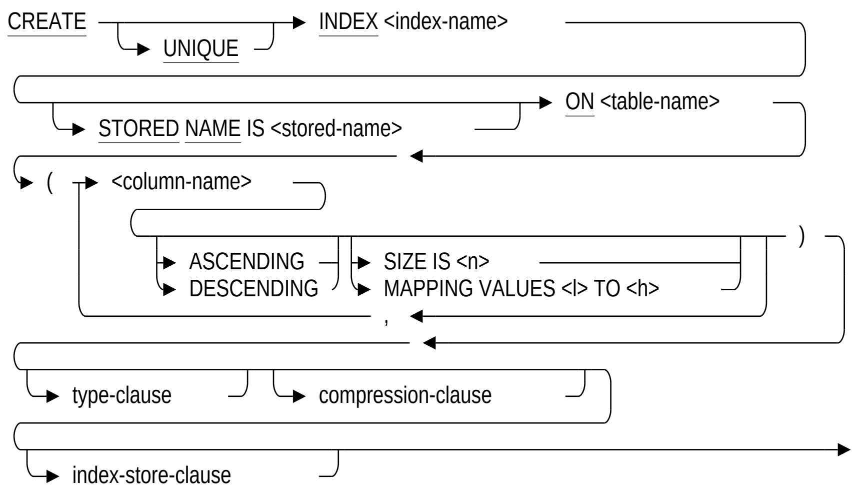
### Environment

You can use the CREATE INDEX statement:

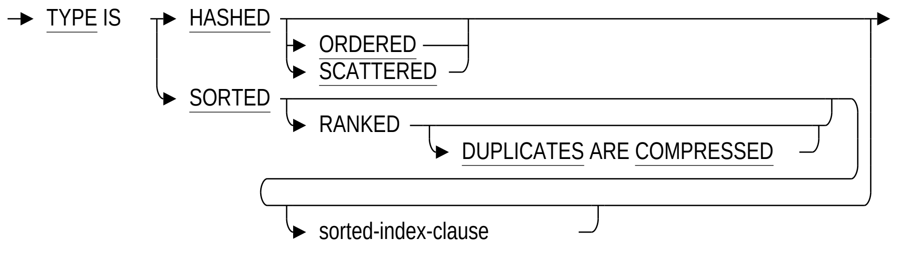
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

## CREATE INDEX Statement

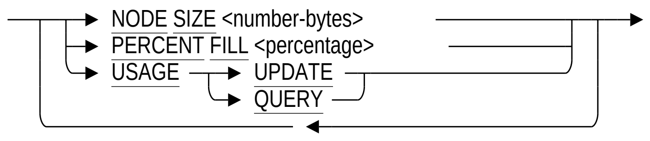
### Format



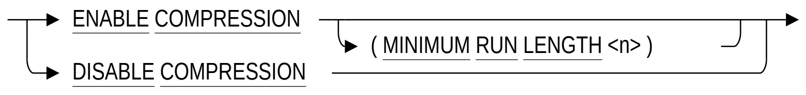
type-clause =



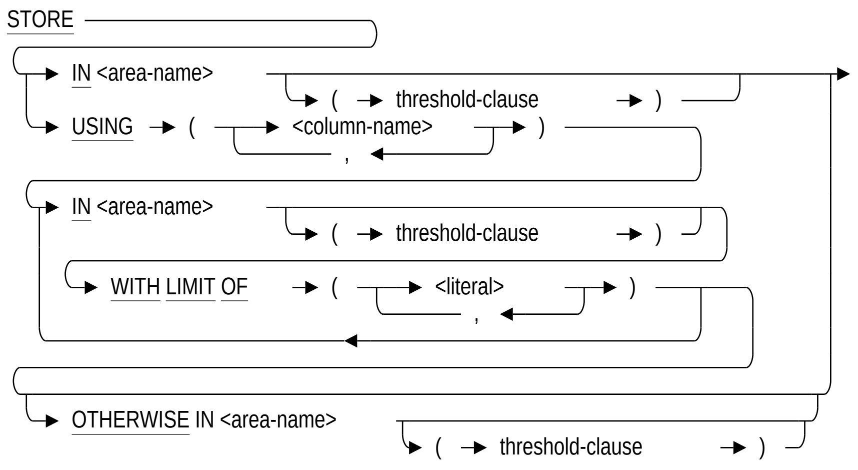
sorted-index-clause =



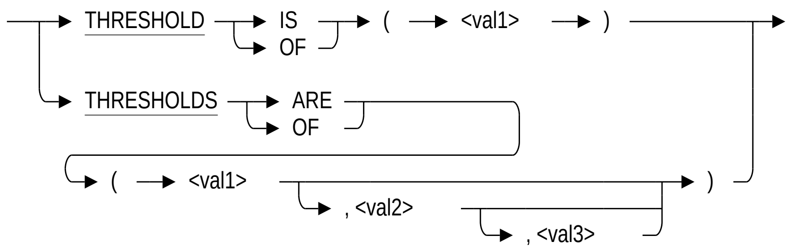
compression-clause =



index-store-clause =



threshold-clause =



## Arguments

### UNIQUE

A keyword that specifies whether or not each value of the index must be unique. If you try to store the same value twice in a column or set of columns that have an index defined as UNIQUE, SQL returns an error message the second time and does not store or modify the row that contains the value. This is true for null values as well as any other value.

If you specify UNIQUE, SQL checks as it executes the CREATE INDEX statement to see if the table already contains duplicate values for the index.

**index-name**
The name of the index. You can use this name to refer to the index in other statements. You must qualify the index name with the authorization identifier if the schema is not the default schema. When choosing a name, specify a valid name. See Section 2.2 for more information on valid, user-supplied names.

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access an index created in a multischema database. The stored name allows you to access multischema definitions using interfaces, such as Oracle RMU, the Oracle Rdb management utility, that do not recognize multiple schemas in one database. You cannot specify a stored name for an index in a database that does not allow multiple schemas. For more information about stored names, see Section 2.2.4.

**table-name**
The name of the table that includes the index. The table must be in the same schema as the index.

**column-name**
The name of the column or columns that make up the index key.

You can create a multisegmented index key by naming two or more columns, which are joined to form the index key. All the columns must be part of the same table. Separate multiple column names with commas.

**Note**

If column-name refers to a column defined as VARCHAR or LONG VARCHAR data type, the size of the column must be less than or equal to 254 characters.

**ASCENDING**
An optional keyword that causes SQL to create ascending index segments. If you omit the ASCENDING or DESCENDING keyword, ascending is the default.

**DESCENDING**
An optional keyword that causes SQL to create descending index segments. If you omit the ASCENDING or DESCENDING keyword, ascending is the default.

**SIZE IS n**

A compression clause for text or varying text index keys that limits the number of characters used for retrieving data. The *n* specifies the number of characters of the key that are used in the index.

**Note**

Although you can create a SIZE IS compressed index and specify the UNIQUE clause, compressing the index key values may make the key values not unique. In this case, the index definition or insert or update statements fail.

**MAPPING VALUES l to h**

A compression clause for all-numeric columns that translates the column values into a compact, encoded form. You can mix mapped and unmapped columns, but the most storage space is gained by building indexes of multiple columns of data type WORD or LONGWORD. Oracle Rdb attempts to compress all such columns into the smallest possible space.

The *l* (low) through *h* (high) specifies the range of integers as the value of the index key.

The valid range of the compressed key (*l* through *h*):

- Cannot be zero
- Is limited to (2**31) – 4 x (10**scale)

  If the value of the key is less than zero or greater than (2**31) – 4 x (10**scale), Oracle Rdb signals an exception.

**TYPE IS HASHED ORDERED**
**TYPE IS HASHED SCATTERED**

Specifies that the index is a **hashed index**. If you specify HASHED, you cannot use the NODE SIZE, PERCENT FILL, or USAGE clauses. You can, however, specify if the data is ORDERED or SCATTERED. SCATTERED is the default.

The TYPE IS HASHED SCATTERED clause is appropriate in situations where data is not evenly distributed across storage areas. This option places a record in a page that is chosen by applying a random algorithm to the index key. As a result, the record distribution pattern is not guaranteed to be even; therefore, some pages may be chosen more often than others. The TYPE IS HASHED SCATTERED clause is the default and is recommended unless your data meets the following criteria for the TYPE IS HASHED ORDERED clause:

- The last column of the index key must be one of the following data types:
  - TINYINT
  - SMALLINT
  - INTEGER
  - BIGINT
  - DATE (both ANSI and VMS)
  - TIME
  - TIMESTAMP
  - INTERVAL
- The index must be ascending.
- The index must not be compressed or have mapping values.

The TYPE IS HASHED ORDERED clause is ideal for applications where the index key values are evenly distributed across a given range. This places a record in a page derived by applying an ordered algorithm to the index key. As a result, the record distribution pattern is guaranteed to follow the index key distribution and is evenly distributed across the pages if the index key values are even. In addition, if you know the range of values, you can size the storage area and pages to minimize overflows. If the index key values are not evenly distributed, use the TYPE IS HASHED SCATTERED clause.

Hashed ordered indexes can be multisegmented. All columns listed, except the last column, can be used in a STORE USING . . . WITH LIMIT clause to partition the data between storage areas.

Hashed indexes must be stored in storage areas created with mixed page format, which means they are valid only in multifile databases.

Hashed indexes provide fast and direct access to specific rows and are effective mainly for queries that specify an exact-match retrieval on a column or columns that are also the key to a hashed index. (For instance, SELECT EMPLOYEE_ID FROM EMPLOYEES WHERE EMPLOYEE_ID = "00126", makes effective use of a hashed index with EMPLOYEE_ID as the index key.)

In a hashed indexing scheme, the index key value is converted mathematically to a relative page number in the storage area of a particular table. A **hash bucket** is a data structure that maintains information about an index key, and a list of internal pointers, called **database keys** or **dbkeys**, to rows that contain the particular value of the index key. To find a row using the hashed index, the database system searches the hash bucket, finds the appropriate dbkey, and then fetches the table row.

Hashed indexes are most effective for random, direct access when the query supplies the entire index key on which the hashed index is defined. For these types of access, I/O operations can be significantly reduced. This is particularly useful for tables with many rows and large indexes. For example, to retrieve a row using a sorted index that is four levels deep, the database system may need to perform five I/O operations. By using hashing, the number of I/O operations is reduced to two, at most.

You can define a hashed index and a sorted index for the same column. Then, depending on the type of query you use, the Oracle Rdb optimizer chooses the appropriate method of retrieval. For example, if your query contains an exact-match retrieval, the optimizer uses hashed index access. If your query contains a range retrieval, the optimizer uses the sorted index. This strategy incurs the additional overhead of maintaining two indexes, therefore, you need to consider the advantages of fast retrieval against the disadvantages of updating two indexes for every change to data.

See the *Oracle Rdb7 Guide to Database Design and Definition* for a detailed discussion of the relative advantages of hashed and sorted indexes.

**TYPE IS SORTED**

Specifies that the index is a sorted, nonranked (B-tree) index. If you omit the TYPE IS clause, SORTED is the default. Sorted indexes improve the performance of queries that compare values using range operators (like BETWEEN and greater than (>)), not exact match operators. (For example, `SELECT EMPLOYEE_ID FROM EMPLOYEES WHERE EMPLOYEE_ID > 200` is a query that specifies a range retrieval and makes effective use of a sorted index.)

You can define a hashed index and a sorted index for the same column. Then, depending on the type of query you use, the Oracle Rdb optimizer chooses the appropriate method of retrieval. For example, if your query contains an exact-match retrieval, the optimizer uses hashed index access. If your query contains a range retrieval, the optimizer uses the sorted index. This strategy incurs the additional overhead of maintaining two indexes; however, you need to consider the advantages of fast retrieval against the disadvantages of updating two indexes for every change to data.

See the *Oracle Rdb7 Guide to Database Design and Definition* for more information on the relative advantages of hashed and sorted indexes.

If you specify a SORTED index, you can optionally specify NODE SIZE, PERCENT FILL, and USAGE clauses that control the characteristics of the nodes in the index.

**Note**

If you define a sorted index for a table that contains no data, the database root node for the index is not created until the first row is stored. When an RMU Verify operation encounters a sorted index without a database root node, it reports the index as empty.

---

**TYPE IS SORTED RANKED**
Specifies that the index is a sorted, ranked (B-tree) index. The ranked B-tree index allows better optimization of queries, particularly queries involving range retrievals. Oracle Rdb can make better estimates of cardinality, reducing disk I/O and lock contention. Oracle Rdb recommends using ranked sorted indexes.

**DUPLICATES ARE COMPRESSED**
Specifies that duplicates are compressed. If a sorted ranked index allows duplicate entries, you can store many more records in a small space when you compress duplicates, therefore, minimizing I/O and increasing performance. Oracle Rdb uses patented technology called byte-aligned bitmap compression to represent the dbkeys for the duplicate entries instead of chaining the duplicate entries together with uncompressed dbkeys.

Duplicates are compressed by default if you specify RANKED without specifying the DUPLICATES ARE COMPRESSED clause.

You cannot use the DUPLICATES ARE COMPRESSED clause when you create nonranked indexes or when you specify the UNIQUE keyword.

See the *Oracle Rdb7 Guide to Database Design and Definition* for more information on sorted ranked B-tree indexes.

**NODE SIZE number-bytes**
The size in bytes of each index node.

The number and level of the resulting index nodes depend on:

- This number-bytes value
- The number and size of the index keys
- The value specified in the PERCENT FILL or USAGE clauses

If you omit the NODE SIZE clause, the default value is:

- 430 bytes if the total index key size is 120 bytes or less
- 860 bytes if the total index key size is more than 120 bytes

The index key size is the number of bytes it takes to represent the column value in the sorted index.

The valid range for a user-specified index node size (in bytes) can be estimated with the following formula:

$$3(key\ size + number\ of\ index\ segments + 11) + 32 \leq node\ size \leq 32767$$

Assuming a key size of 1 and 1 segment, the minimum acceptable node size value is 71 bytes.

The formula for the minimum node-size value is based on the following rationale:

- $3$

  Ensures that 3 entries always fit in an index node, which further ensures that a perfect binary tree does not result. With index key compression, more than 3 entries frequently fit into this minimally sized node.

- $key\ size$

  The number of bytes required to represent the needed columns in the sorted index. Generally this is the number of storage bytes, plus 1 byte for each column (the 1 is the "null byte").

- $number\ of\ index\ segments$

  The number of segments (columns) defined in the key.

- $11$

  The maximum number of overhead bytes per index key within a node: 1 byte for "how many bytes of last entry are prefixed to this one"; 1 byte for "how many bytes are in this entry"; and 9 bytes for a database key (dbkey) that cannot be compressed.

- $32$

  The index node overhead.

Although the maximum index node size is 65,000 bytes, a practical upper limit is the database page size in bytes minus the overhead of the page and the space needed to store one complete row. For a default page of 2 blocks (1024 bytes), 860 bytes is a reasonable maximum size.

If you do not specify the NODE SIZE clause, Oracle Rdb creates the nodes with default sizes of:

- 430 bytes when:

$$3(key\ size + number\ of\ index\ segments + 11) + 32) < 430\ bytes$$

- 860 bytes when:

$$3(key\ size + number\ of\ index\ segments + 11) + 32) \geq 430\ bytes$$

For example, if the key size is 121 bytes and the user specified a value lower than the minimum of $3(121+1+11)+32$, or 431 bytes, then Oracle Rdb supplies the value of 860 bytes.

If you specify a node size that is less than the required size, an error is returned and the index is not created.

**PERCENT FILL percentage**
Specifies the initial fullness percentage for each node in the index structure being changed. The valid range is 1 percent to 100 percent. The default is 70 percent.

Both the PERCENT FILL and USAGE clauses specify how full an index node should be initially. Specify either PERCENT FILL or USAGE, but not both. However, if you specify both, SQL uses the last clause specified.

**USAGE UPDATE**
**USAGE QUERY**
Specifies a PERCENT FILL value appropriate for update- or query-intensive applications. The USAGE UPDATE clause sets the PERCENT FILL value at 70 percent. The USAGE QUERY clause sets the PERCENT FILL value at 100 percent.

Supplying the PERCENT FILL and USAGE clauses is allowed in the syntax but is semantically meaningless. Specify either PERCENT FILL or USAGE, but not both. However, if you specify both, SQL uses the last clause specified.

**ENABLE COMPRESSION**
Specifies that sorted and hashed indexes are stored in a compressed form.

If compression is enabled, Oracle Rdb uses run-length compression to compress a sequence of space characters (octets) from text data types and binary zeros from nontext data types. Different character sets have different representations of the space character. Oracle Rdb compresses the representation of the space character for the character sets of the columns comprising the index values.

You cannot disable index compression using the ALTER INDEX statement once you specified the ENABLE COMPRESSION clause of the CREATE INDEX statement.

For more information on compressed indexes, see the *Oracle Rdb7 Guide to Database Design and Definition*.

#### MINIMUM RUN LENGTH

Specifies the minimum length of the sequence that Oracle Rdb should compress. You cannot alter this value once you set it.

If you specify MINIMUM RUN LENGTH 2, Oracle Rdb compresses sequences of two or more spaces or of two or more binary zeros for single-octet character sets, and compresses one space or one binary zero for multi-octet character sets. As it compresses the sequences, Oracle Rdb replaces the sequence with the value of the minimum run length plus 1 byte. If many of the index values contain one space between characters in addition to trailing spaces, use a minimum run length of 2, so that you do not inadvertently expand the index beyond the 255-byte limit. If you inadvertently expand the index beyond 255 bytes during index creation, Oracle Rdb returns a warning message.

The default minimum run length value is 2. Valid values for the minimum run length range from 1 to 127. Oracle Rdb determines which characters are compressed.

#### DISABLE COMPRESSION

Disables compression indexes.

If compression is disabled, no form of compression is used for hashed indexes, and prefix compression or suffix compression is used for sorted indexes. **Prefix compression** is the compression of the first bytes of an index key that are common in consecutive index keys. Prefix compression saves space by not storing these common bytes of information. Conversely, **suffix compression** is the compression of the last bytes from adjacent index keys. These bytes are not necessary to guarantee uniqueness.

You cannot enable index compression using the ALTER INDEX statement once you specified the DISABLE COMPRESSION clause of the CREATE INDEX statement.

Index compression is disabled by default.

#### index-store-clause

A storage map definition for the index. You can specify a store clause for indexes in a multifile schema only. The STORE clause in a CREATE INDEX statement allows you to specify which storage area files are used to store the index entries:

- All index entries can be associated with a single storage area.
- Index entries can be systematically distributed, or partitioned, among several storage areas by specifying upper limits on the values for a key in a particular storage area.

If you omit the storage map definition, the default is to store all the entries for an index in the main RDB$SYSTEM storage area.

You should define a storage area for an index that matches the storage map for the table with which it is associated.

In particular, under the following conditions, the database system stores the index entry for a row on or near the same data page that contains the actual row:

- The storage areas for a table have a mixed page format.
- You specify an identical store clause for the index as exists in the storage map for the table.
- The storage map for the table also names the index in the PLACEMENT VIA INDEX clause.

Such coincidental clustering of indexes and rows can reduce I/O operations. With hashed indexes and coincidental clustering, the database system can retrieve rows for exact-match queries in one I/O operation.

For sorted indexes, specifying an identical storage map reduces I/O contention on index nodes.

**STORE IN area-name**
Associates the index directly with a single storage area. All entries in the index are stored in the area you specify.

**threshold-clause**
Specifies one, two, or three default threshold values for logical areas that contain the index in storage areas with uniform page formats. By setting threshold values, you can make sure that Oracle Rdb does not overlook a page with sufficient space to store compressed data. The threshold values (val1, val2, and val3) represent a fullness percentage on a data page and establish three possible ranges of guaranteed free space on the data pages. For more information about logical area thresholds, see the CREATE STORAGE MAP Statement.

If you use data compression, you should use logical area thresholds to obtain optimum storage performance.

You cannot specify the thresholds for the storage map attribute for any area that is a mixed page format. If you have a mixed page format, set the thresholds for the storage area using the ADD STORAGE AREA or CREATE STORAGE AREA clause of the ALTER DATABASE, CREATE DATABASE, or IMPORT statements.

For more information about SPAM pages, see the *Oracle Rdb7 Guide to Database Design and Definition*.

**STORE USING (column-name-list)**
Specifies columns whose values are used as limits for partitioning the index across multiple storage areas. You cannot name columns not specified as index key segments.

If the index key is multisegmented, you can include some or all the columns that are joined to form the index key. You must specify the columns in the order in which they were specified when the index key was defined. If you include only a subset of the columns from the multisegmented index, you must include the first column of the segment.

For example, if a CREATE INDEX statement specifies a multisegmented index based on the columns LAST_NAME, FIRST_NAME, and MIDDLE_INITIAL, the STORE USING clause allows the following combinations of columns:

- LAST_NAME
- LAST_NAME and FIRST_NAME
- LAST_NAME, FIRST_NAME, and MIDDLE_INITIAL

You cannot specify only FIRST_NAME, or FIRST_NAME and MIDDLE_INITIAL, in the STORE USING clause.

The database system uses the values of the columns specified in the STORE USING clause as a key to determine in which storage area an index entry associated with a new table row belongs.

**IN area-name**
Associates the index directly with a single storage area. All entries in the index are stored in the area you specify.

**WITH LIMIT OF (literal-list)**
Specifies the highest value for the index key that resides in a particular storage area if ASCENDING is defined. If DESCENDING is defined, the lowest value

is specified for the index key that resides in a particular storage are. For multicolumn index keys, specify a literal value for each column.

The number of literals in the list must be the same as the number of columns in the USING clause. Repeat this clause to partition the entries of an index among multiple storage areas. The data type of the literals must agree with the data type of the column. For character columns, enclose the literals in single quotation marks.

If the columns in the row from which the key is defined are changed, the index entry is not moved to a different storage area.

If you are creating a multisegmented index using multisegmented keys and the STORE USING . . . WITH LIMIT clause, and if the values for the first key are all the same, then set the limit for the first key at that value. This ensures that the value of the second key determines the storage area in which each row is stored.

**OTHERWISE IN area-name**
For partitioned storage maps only, specifies the storage area that is used as the overflow partition. An **overflow partition** is a storage area that holds any values that are higher than those specified in the last WITH LIMIT TO clause. An overflow partition holds those values that “overflow” the partitions that have specified limits.

## Usage Notes

- When the CREATE INDEX statement executes, SQL adds the index definition to the physical database. If you have declared the schema with the PATHNAME argument, the index definition is also added to the repository.
- You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with characteristics specified in the most recent DECLARE TRANSACTION statement.
- You can create indexes at the same time other users are creating indexes, even if the indexes are on the same table. To allow concurrent index definition on the same table, use the SHARED DATA DEFINITION clause of the SET TRANSACTION statement. For more information, see the SET TRANSACTION Statement.

- The CREATE INDEX statement fails when both of the following circumstances are true:
  - The database to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The database was attached using the FILENAME argument.

  Under these circumstances, the statement fails with the following error when you issue it:

  ```
  %RDB-E-NO_META_UPDATE, metadata update failed
  -RDMS-F-CDDISREQ, CDD required for metadata updates is not being maintained
  ```
- You cannot execute the CREATE INDEX statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on the RDB$SYSTEM storage area.
- If the character length is specified in octets, which is the default, the size specified in the compression clause is also in octets.
- If the character length is specified in characters, the size specified in the compression clause is also in characters.
- A CREATE INDEX statement following a DROP INDEX statement does not reuse the space made available by the previous statement, as shown in the following MF_PERSONNEL database example. As a result, when you display the page numbers used, they are different.

  ```
  SQL> CREATE INDEX INDEX1 ON EMPLOYEES (LAST_NAME) STORE IN RDB$SYSTEM;
  SQL> COMMIT;
  SQL> $ RMU/DUMP/LAREA=INDEX1 MF_PERSONNEL
  SQL> DROP INDEX INDEX1;
  SQL> COMMIT;
  SQL> CREATE INDEX INDEX1 ON EMPLOYEES (LAST_NAME) STORE IN RDB$SYSTEM;
  SQL> COMMIT;
  SQL> $ RMU/DUMP/LAREA=INDEX1 MF_PERSONNEL
  ```

  Currently Oracle Rdb does not reclaim clumps belonging to a table or index until the process that deleted that clump is disconnected using the DISCONNECT or FINISH statement.

  Technically, the restriction could be that the clump cannot be reclaimed until the transaction that deleted the page is committed.

  For more information on this restriction, see the *Oracle Rdb7 Guide to Database Design and Definition*.

- If you include a comment after the STORE clause in the CREATE INDEX statement, the comment is included in the index and table information.

  ```
  SQL> CREATE DATABASE FILENAME t
  cont> CREATE STORAGE AREA a
  cont> CREATE STORAGE AREA b
  cont> CREATE TABLE T1 (A CHAR(15), B INT)
  cont> CREATE INDEX I1 ON T1 (A) STORE IN b
  cont> -- TEST CREATE INDEX
  cont> ;
  SQL> SHOW INDEX I1
  Indexes on table T1:
  I1                              with column A
    Duplicates are allowed
    Type is Sorted
  Store clause:          STORE IN b
                      -- TEST CREATE INDEX
  ```

  Use caution when using the STORE clause.

- The maximum length of an index key is 255 bytes. Because Oracle Rdb generates fixed-length index keys, this constraint is checked at the time the index is defined. If you attempt to define an index with a key larger than 255 characters, you get the following error message:

  ```
  SQL> ATTACH 'FILENAME mf_personnel';
  SQL> CREATE TABLE TEST_TAB (TEST_COL CHAR (256));
  SQL> CREATE INDEX MY_INDEX ON TEST_TAB (TEST_COL);
  %RDB-E-NO_META_UPDATE, metadata update failed
  -RDMS-F-INDTOOBIG, requested index is too big
  ```

- Database designers should be aware of the following optimizer restrictions concerning references to fields with the COLLATING SEQUENCE attribute or fields whose data type is VARCHAR (VARYING STRING). These restrictions affect performance with respect to I/O operations.

  The optimizer Index Only Retrieval and Key-Only Boolean strategies are disabled if any field in the index has a collating sequence defined, or is a VARYING STRING field. These two retrieval strategies require Oracle Rdb to return data stored in the index node or perform comparisons based on the index node key fields, thus saving I/O operations to the data record. However, the original user data cannot be reconstructed from the encoded index if these attributes are used. Therefore, the optimizer forces a Retrieval by Index strategy instead, which requires I/O operations to the data record.

These restrictions may affect the choice of data type for fields to be used in indexes. For example, PRODUCT_ID, which has a data type of CHAR(20), is part of an index P_INDEX. A query that uses STARTING WITH against PRODUCT_ID allows the user to enter a partial product code. It then fetches the matched PRODUCT_ID field for display to the user, but does not fetch any other fields. This query would normally be optimized to reference the index PRODUCT_ID_IX only (that is, using an Index Only Retrieval strategy). However, if the field was defined as VARCHAR(20), the optimizer would be required to reference the data record to fetch the PRODUCT_ID. This will add some extra I/O operations to the translation query. Therefore, CHAR (TEXT) data type may be preferable to VARCHAR (VARYING STRING) if the field is involved in index retrieval.

The following example demonstrates this simple case. The optimizer strategy is displayed when the RDMS$DEBUG_FLAGS logical name or RDB_DEBUG_FLAGS configuration parameter is set to “S” has been inserted after each query. You can also use the SET FLAGS statement to set the debug flags.

```
SQL> SHOW TABLE PRODUCTS
Columns for table PRODUCTS:
Column Name                     Data Type        Domain
-----------                     ---------        ------
PRODUCT_ID_V                    VARCHAR(20)      PRODUCT_ID_V
PRODUCT_ID_T                    CHAR(20)         PRODUCT_ID_T
   .
   .
   .
Indexes on table PRODUCTS:
P_INDEX_T                       with column PRODUCT_ID_T
                                duplicates are allowed
                                type is sorted

P_INDEX_V                       with column PRODUCT_ID_V
                                duplicates are allowed
                                type is sorted
   .
   .
   .
SQL>
SQL> SELECT     PRODUCT_ID_T
cont> FROM      PRODUCTS
cont> WHERE     PRODUCT_ID_T STARTING WITH 'AAA';
    Conjunct        Get     Index only retrieval
    Retrieval by index of relation PRODUCTS        Index name P_INDEX_T
    00000001 Segments in low Ikey   00000001 Segments in high Ikey
0 rows selected
```

```
SQL>
SQL> SELECT     PRODUCT_ID_V
cont> FROM      PRODUCTS
cont> WHERE     PRODUCT_ID_V STARTING WITH 'AAA';
    Conjunct        Get     Retrieval by index of relation PRODUCTS
    Index name P_INDEX_V    00000001 Segments in low Ikey
    00000001 Segments in high Ikey
0 rows selected
SQL>
SQL> COMMIT;
```

**Note**

Most queries use indexes as a fast access method to reference rows (records) of data, so an I/O operation to the data record is normally required.

---

- If it is unlikely that you store values greater than a specific range, you can omit the OTHERWISE clause. This lets you quickly add new partitions without reorganizing the storage areas. To add new partitions, use the ALTER INDEX statement. For example:

  ```
  SQL> ALTER INDEX EMP_HASH_INDEX
  cont>      STORE USING (EMPLOYEE_ID)
  cont>            IN PERSONNEL_1 WITH LIMIT OF ('00399')
  cont>            IN PERSONNEL_2 WITH LIMIT OF ('00699')
  cont>            IN PERSONNEL_3 WITH LIMIT OF ('10000')
  cont>            IN PERSONNEL_4 WITH LIMIT OF ('10399');
  SQL>
  ```

  Because Oracle Rdb does not have to move or reorganize data, you can quickly alter indexes that do not contain overflow partitions.

  For more information, see the *Oracle Rdb7 Guide to Database Design and Definition* and the *Oracle Rdb7 Guide to Database Performance and Tuning*.

- If you attempt to insert values that are out of range of the index, you receive an error similar to the following:

  ```
  %RDMS-E-EXCMAPLIMIT, exceeded limit on last partition in storage map for
     EMPLOYEES
  ```

  Your applications should include code that handles this type of error.

- For effective parallel sort index builds against the same database table, the index name of each index being built concurrently must be unique within the first 27 characters. Failure to specify a unique name creates only one sort index because each index build requests the same name lock prior to the start of each index build.

- System record logical areas are created at the same time as the corresponding mixed format storage area. Oracle Rdb checks for duplicate system record logical areas and prevents parallel index definers from creating additional system record logical areas. If you attempt to create a system record logical area that already exists, Oracle Rdb generates an error similar to the following:

  ```
  %RDB-E-NO_META_UPDATE, metadata update failed
   -RDMS-E-DUPSYSREC, Cannot create duplicate system record in "INDEX_AREA_1"
  ```

- You cannot create a unique index and specify duplicates compression for a sorted ranked B-tree index. For example:

  ```
  SQL> CREATE UNIQUE INDEX test_ndx
  cont> ON job_history (employee_id)
  cont> TYPE IS SORTED RANKED
  cont> DUPLICATES ARE COMPRESSED;
  %SQL-F-UNIQNODUP, The index TEST_NDX cannot be unique and have a duplicates
  clause
  ```

## Examples

Example 1: Creating a simple table index

This statement names the index (EMP_EMPLOYEE_ID) and names the column to serve as the index key (EMPLOYEE_ID).

The UNIQUE argument causes SQL to return an error message if a user tries to store an identification number that is already assigned.

```
SQL> CREATE UNIQUE INDEX EMP_EMPLOYEE_ID ON EMPLOYEES
cont>    (EMPLOYEE_ID);
```

Example 2: Creating an index with descending index segments

This statement names the index (EMP_EMPLOYEE_ID) and names the column to serve as the descending index key (EMPLOYEE_ID DESCENDING).

The DESCENDING keyword causes the keys to be sorted in descending order. If you do not specify DESCENDING or ASCENDING, SQL sorts the keys in ascending order.

```
SQL> CREATE UNIQUE INDEX EMP_EMPLOYEE_ID ON EMPLOYEES
cont>    (EMPLOYEE_ID DESCENDING);
```

Example 3: Creating a multisegmented index

```
SQL> CREATE INDEX EMP_FULL_NAME ON EMPLOYEES
cont>     (LAST_NAME,
cont>      FIRST_NAME,
cont>      MIDDLE_INITIAL);
```

This statement names three columns to be used in the index EMP_FULL_NAME. SQL concatenates these three columns to make the multisegmented index.

Example 4: Creating a compressed numeric index

```
SQL> CREATE INDEX YEAR1_IND ON DEGREES
cont> (YEAR_GIVEN ASCENDING MAPPING VALUES 1950 TO 1970);
```

This statement creates ascending index segments for the YEAR_GIVEN column in the DEGREES table, compressing the year values.

Example 5: Creating a compressed text index

```
SQL> CREATE INDEX COL_NAME_IND ON COLLEGES
cont> (COLLEGE_NAME SIZE IS 20);
```

This statement creates a compressed index, COL_NAME_IND, on the COLLEGES table so that the number of octets from the COLLEGE_NAME column that are used as a key cannot exceed 20 octets.

Example 6: Creating an index in a uniform storage area with thresholds.

```
SQL> ALTER DATABASE FILENAME mf_personnel
cont>  ADD STORAGE AREA UNIFORM1 PAGE FORMAT IS UNIFORM;
SQL> ALTER DATABASE FILENAME mf_personnel
cont>  ADD STORAGE AREA UNIFORM2 PAGE FORMAT IS UNIFORM;
SQL> ATTACH 'FILENAME mf_personnel';
SQL> CREATE UNIQUE INDEX EMP_THRESHOLDS ON EMPLOYEES (EMPLOYEE_ID)
cont>   TYPE IS SORTED
cont>   STORE USING (EMPLOYEE_ID)
cont>        IN RDB$SYSTEM (THRESHOLDS ARE (60,75,90))
cont>                     WITH LIMIT OF ('00200')
cont>        IN UNIFORM1 (THRESHOLD IS (65))
cont>                     WITH LIMIT OF ('00400')
cont>        OTHERWISE IN UNIFORM2
cont>                   (THRESHOLD OF (90));
SQL> --
SQL> SHOW INDEX EMP_THRESHOLDS
Indexes on table EMPLOYEES:
EMP_THRESHOLDS                  with column EMPLOYEE_ID
  No Duplicates allowed
  Type is Sorted
Store clause:          STORE USING (EMPLOYEE_ID)
                            IN RDB$SYSTEM (THRESHOLDS ARE (60,75,90))
                              WITH LIMIT OF ('00200')
                            IN UNIFORM1 (THRESHOLD IS (65))
                              WITH LIMIT OF ('00400')
                            OTHERWISE IN UNIFORM2
                              (THRESHOLD OF (90))
```

This statement uses the STORE clause to partition the index into different uniform page format storage areas and apply thresholds.

In Examples 7 and 8, the table COLOURS in the database MIA_CHAR_SET is defined as:

```
SQL> CREATE TABLE COLOURS
cont>        (ENGLISH   MCS_DOM,
cont>         FRENCH    MCS_DOM,
cont>         JAPANESE  KANJI_DOM,
cont>         ROMAJI    DEC_KANJI_DOM,
cont>         KATAKANA  KATAKANA_DOM,
cont>         HINDI     HINDI_DOM,
cont>         GREEK     GREEK_DOM,
cont>         ARABIC    ARABIC_DOM,
cont>         RUSSIAN   RUSSIAN_DOM);
```

Example 7: Creating a simple table index using the octets character length, which is the default

```
SQL> SET CHARACTER LENGTH 'OCTETS';
SQL> CREATE INDEX COLOUR_INDEX ON COLOURS (JAPANESE SIZE IS 4)
cont> TYPE IS SORTED;
SQL> SHOW INDEX COLOUR_INDEX;
Indexes on table COLOURS:
COLOUR_INDEX                    with column JAPANESE
                                size of index key is 4 octets
  Duplicates are allowed
  Type is Sorted
```

The previous statement creates a compressed index key of 4 octets.

Example 8: Creating an index using the CHARACTERS character length

```
SQL> SET CHARACTER LENGTH 'CHARACTERS';
SQL> CREATE INDEX COLOUR_INDEX_2 ON COLOURS (JAPANESE SIZE IS 4)
cont> TYPE IS SORTED;
SQL> SHOW INDEX COLOUR_INDEX_2;
Indexes on table COLOURS:
COLOUR_INDEX_2                  with column JAPANESE
                                size of index key is 4 characters
  Duplicates are allowed
  Type is Sorted
```

The previous statement creates a compressed index key of 4 characters.

Example 9: Creating an index that enables compression

The following example shows how to create an index and enable compression with a minimum run length of 2:

```
SQL> CREATE INDEX EMP_NDX ON EMPLOYEES
cont> (EMPLOYEE_ID SIZE IS 4)
cont>  ENABLE COMPRESSION (MINIMUM RUN LENGTH 2);
SQL> SHOW INDEX EMP_NDX;
Indexes on table EMPLOYEES:
EMP_NDX                         with column EMPLOYEE_ID
                                size of index key is 4
  Duplicates are allowed
  Type is Sorted
  Compression is ENABLED  (Minimum run length  2)
```

## CREATE MODULE Statement

Defines a module as an object in an Oracle Rdb database. Stored with the module are its functions and procedures. A function or procedure that resides with the data in a database is called a **stored function** or **stored procedure**. Likewise, a module stored in a database is called a **stored module**. A **stored routine** refers to either a stored procedure or stored function.

You invoke a stored procedure with the CALL statement from a simple statement procedure in embedded SQL, SQL module language, or interactive SQL or with the CALL statement from within a compound statement.

You invoke a stored function by specifying the function name in a value expression.

SQL uses the concept of a module as its mechanism for storing, showing, deleting, and granting and revoking privileges on stored routines within a database. This means you cannot store, delete, or grant and revoke privileges on individual stored routines. Should you need to remove a stored routine, use the DROP FUNCTION routine-name CASCADE or DROP PROCEDURE routine-name CASCADE syntax.

In general, SQL operates on modules, not stored routines. However, there are a few exceptions: DROP FUNCTION, DROP PROCEDURE, SHOW FUNCTION, SHOW PROCEDURE, and CALL. The SHOW FUNCTION statement displays information about functions. The SHOW PROCEDURE statement displays individual procedures in a stored module. The CALL statement can invoke only a single stored procedure.

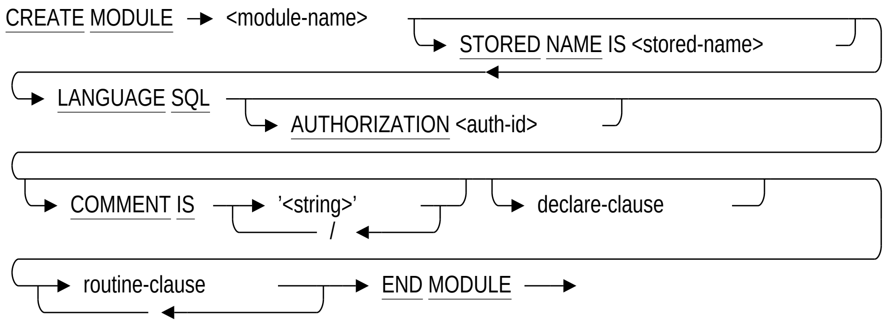
### Environment

You can use the CREATE MODULE statement in a simple statement procedure:

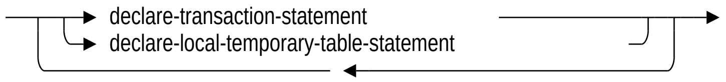
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

## CREATE MODULE Statement

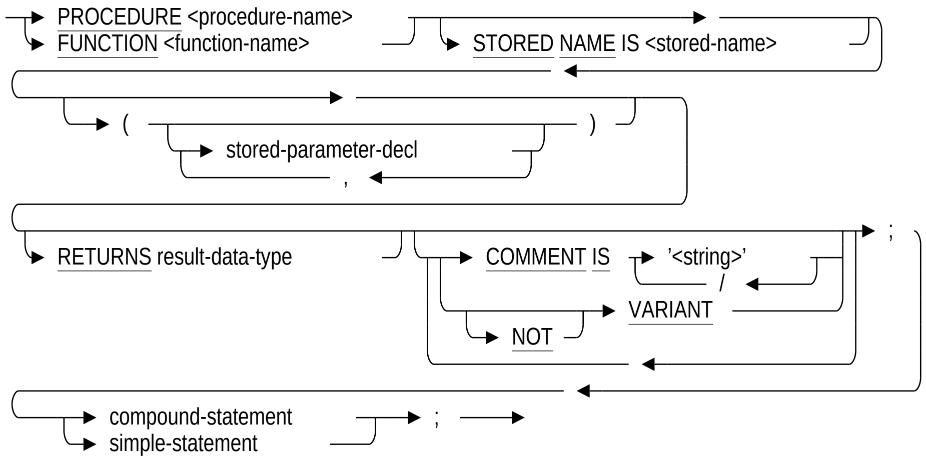
### Format



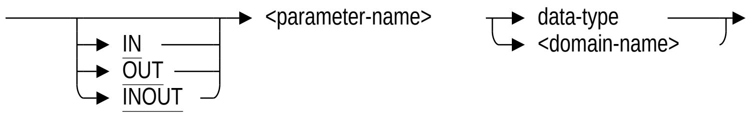
declare-clause =


declare-transaction-statement
declare-local-temporary-table-statement

routine-clause =



stored-parameter-decl =



## Arguments

**module-name**
A user-supplied name that you assign to a module.

See Section 2.2 for more information on user-supplied names.

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access a module created in a multischema database.

**LANGUAGE SQL**
The LANGUAGE keyword and the SQL argument signify that the procedures in a module are to be invoked by SQL statements, not a host language program.

With unstored procedures, the LANGUAGE keyword specifies the name of a host language; this identifies the host language in which the program calling a module's procedures is written.

**AUTHORIZATION auth-id**
A name that identifies the definer of a module and is used to perform privilege validation for the module.

See the Usage Notes for more information about privilege validation and Section 2.2.5 for information about using authorization identifiers.

**declare-transaction-statement**
Declares a transaction for the module. See the DECLARE TRANSACTION Statement for more information.

**declare-local-temporary-table-statement**
Declares a local temporary table for the module. See the DECLARE LOCAL TEMPORARY TABLE Statement for more information.

**COMMENT IS 'string'**
Adds a comment about the module. SQL displays the text of the comment when it executes a SHOW MODULE statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

**routine-clause**
The definition of a stored function or stored procedure created in a module.

**PROCEDURE procedure-name**
**FUNCTION function-name**
A user-supplied name that you give to a stored routine in a module. The name you specify for a stored routine must be unique within the database definition.

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access a procedure or function created in a stored module.

**stored-parameter-decl**
Specifies the parameters and parameter modes used in a stored routine.

**IN**
**OUT**
**INOUT parameter-name**
Specifies the parameter modes used in a stored routine.

The IN parameter names the parameter that is read into the stored routine, however it is never set. The OUT parameter names the parameter into which data is being sent. The OUT parameter is set, but never read. The INOUT parameter names a parameter that inputs data (is read) as well as receives data (is set). The INOUT parameter is a parameter that is modified.

The IN parameter is the only mode allowed for stored functions.

Each parameter name must be unique within the stored routine.

**data-type**
A valid SQL data type. Specifying an explicit data type is an alternative to specifying a domain name. See Section 2.3 for more information on data types.

**domain-name**
The name of a domain created in a CREATE DOMAIN statement. For more information about domains, see the CREATE DOMAIN Statement.

**RETURNS result-data-type**
Specifies the data type or domain of the result of the function invocation. This clause is only valid when defining a stored function. You can only use the RETURNS clause when defining a stored function. You cannot use the RETURNS clause when defining a stored procedure.

**COMMENT IS 'string'**
Adds a comment about the stored routine. SQL displays the text of the comment when it executes a SHOW PROCEDURE or SHOW FUNCTION statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

**VARIANT**
**NOT VARIANT**
The VARIANT or NOT VARIANT clause controls the evaluation of a stored function in the scope of a query:

- VARIANT

  Specifying the VARIANT clause forces evaluation of corresponding functions (in scope of a single query) every time the function appears. If a function can return a different result each time it is invoked, you should use the VARIANT clause.

- NOT VARIANT

  Specifying the NOT VARIANT clause can result in a single evaluation of corresponding function expressions (in scope of a single query), and the resulting value is used in all occurrences of the corresponding function expression. When you use the NOT VARIANT clause, Oracle Rdb evaluates whether or not to invoke the function each time it is used.

  For example:

  ```
  SELECT * FROM T1 WHERE F1() > 0 AND F1() < 20;
  ```

  If you define the F1 function as NOT VARIANT, the function F1() may be evaluated just once depending on the optimizer. If you define the F1 function as VARIANT, the function F1() is evaluated twice.

  NOT VARIANT is the default.

The VARIANT or NOT VARIANT clause is not allowed on stored procedure definitions.

**compound-statement**
Allows you to include more than one SQL statement in a stored routine. See the Compound Statement for more information.

**simple-statement**
Allows you to include one SQL statement in a stored routine. See the Simple Statement for more information.

If you are defining a stored function, the simple statement must be the RETURNS clause.

## Usage Notes

- You must have CREATE privilege on the database to create modules in that database.
- When the module definition contains an AUTHORIZATION clause, authorization validation checks that the authorization identifier that you specify is a valid user name. On OpenVMS, a valid user name can be an OpenVMS rights identifier as well as a user name.

  Then, it validates privileges for this authorization ID for all objects referred to in the module. Such a module is a definer's rights module because the system executes the module procedures under the authorization ID of the module definer.

  Definer's rights modules greatly reduce the number of privileges that need to be granted in a database because only the module definer requires privileges on the objects referenced in the module. All other users executing procedures in the module require an EXECUTE privilege.

  An invoker's rights module is a stored module that does not contain an AUTHORIZATION clause. At run time, the identifier of the user that invokes a procedure contained in the module is used to perform privilege validation for all objects referenced by the module.
- Before any invoker's rights routines are called, CURRENT_USER is established as identical to the SESSION_USER. As each routine is called it either inherits this value from the caller or, in the case of a definer's rights routine, it is derived from the module AUTHORIZATION clause. Therefore, CURRENT_USER returns the authorization of the last definer's rights routine in the call chain.
- You invoke a stored procedure using the CALL statement. You can also invoke a stored procedure from a compound statement or from another stored procedure.

  See the CALL Statement for Simple Statements and the CALL Statement for Compound Statements for more information on invoking a stored procedure.
- The following highlight some differences between stored and nonstored procedures:
  - Stored procedures allow null values to be passed by the parameters; nonstored procedures must use indicator variables.

  - You cannot declare a status parameter such as SQLCODE, SQLSTATE, or SQLCA in a stored procedure; you must declare a status parameter for nonstored procedures.
  - All SQL data types are allowed in stored procedures. Depending on the host language used, some data types are not allowed in nonstored procedures.
  - Stored and nonstored module names must be unique from each other. If you attempt to invoke a stored module while a nonstored module with the same name is active, you receive the following error:

    ```
    %RDB-E-IMP_EXC, facility-specific limit exceeded
    -RDMS-E-MODEXTS, there is another module named SALARY_ROUTINES in
    this database
    ```

- Stored routine names must be unique from other stored and external routines.
- If you alter or delete certain SQL elements, you can cause SQL to invalidate the stored routines that use those elements.

  In general, any DROP statement that is restricted does not affect stored routine validation. A statement with the RESTRICT keyword prevents the deletion of any objects that have any stored routine dependencies. Drop cascade operations execute successfully, but cause stored routine invalidation.

  Table 6–3 shows which statements can cause SQL to invalidate stored routines.

**Table 6–3 ALTER and DROP Statements Causing or Not Causing Stored Routine Invalidation**

| Object Type | SQL Statement | Does This Statement Fail? | Stored Routine Invalidated? | Dependency Type[1] |
|---|---|---|---|---|
| Column | ALTER TABLE DROP COLUMN | Yes | No | SR |
| | ALTER TABLE ADD COLUMN | No | Yes | LS |
| | ALTER TABLE ADD COLUMN | No | No | SR |
| Constraint | ALTER TABLE DROP CONSTRAINT | Yes | No | SR |
| | ALTER TABLE ADD CONSTRAINT | No | No | SR or DE |
| Domain | ALTER DOMAIN (in parameter list) | Yes | No | Does not apply[2] |
| | ALTER DOMAIN (in procedure block) | No | No | SR[3] |
| | DROP DOMAIN | Yes | No | SR or SM |
| Function | DROP FUNCTION RESTRICT | Yes | No | SR |
| | DROP FUNCTION CASCADE | No | Yes | SR |
| Module | DROP MODULE RESTRICT | Yes | No | SR |
| | DROP MODULE CASCADE | No | Yes | SR |
| Procedure | DROP PROCEDURE RESTRICT | Yes | No | SR |
| | DROP PROCEDURE CASCADE | No | Yes | SR |
| Table | DROP TABLE RESTRICT | Yes | No | SR, LS, DR, or SM |
| | DROP TABLE CASCADE | No | Yes | SR or LS |
| View | DROP VIEW RESTRICT | Yes | No | SR, LS, or DR |
| | DROP VIEW CASCADE | No | Yes | SR or LS |

[1]Dependency types: **DE** (default evaluating), **DR** (default reserving), **LS** (language semantics), **SR** (stored routine), **SM** (stored module).

[2]Oracle Rdb keeps this domain parameter list dependency in RDB$PARAMETERS, not in RDB$INTERRELATIONS.

[3]Oracle Rdb stores routine dependencies in RDB$INTERRELATIONS when a domain exists in a stored routine block.

- The maximum length for each string literal in a comment is 1,024 characters.
- You cannot specify the COMMIT, ROLLBACK, or SET TRANSATION statements within a stored function.
- You can invoke a stored function anywhere a value expression is allowed.

- You must specify a value expression for each parameter specified in the stored function definition.
- There is no limit set by Oracle Rdb as to the depth of nesting allowed for stored routines. Memory and stack size are the only constraints to consider here.
- Stored routines can reference any other previously defined stored routines in the module.
- The TRACE flag can be used from any stored routine. However, if you enable the TRACE flag from within SQL, you must do so before the stored routine is invoked.

## Examples

Example 1: Creating a stored module and stored procedure

The following example shows how to create a stored module and stored procedure using interactive SQL:

```
SQL> CREATE MODULE testmod LANGUAGE SQL
cont> PROCEDURE testproc;
cont> BEGIN
cont> COMMIT;
cont> END ;
cont> END MODULE;
SQL> SHOW MODULE testmod
 Module name is: TESTMOD
 Source:
 TESTMOD LANGUAGE SQL
 Owner is:
 Module ID is: 1
```

Example 2: Creating a stored module with module SQL

The following code segment shows how to create a stored module as part of a procedure in a nonstored module:

```
PROCEDURE create_them
    SQLCODE;
    CREATE MODULE my LANGUAGE SQL AUTHORIZATION smith
    PROCEDURE p1 ( :x CHAR(5) );
        BEGIN
        INSERT INTO s (snum) VALUES (:x);
        END;
    PROCEDURE p2 ( :y SMALLINT );
        BEGIN
        SELECT STATUS INTO :y FROM s LIMIT TO 1 ROW;
        END;
    PROCEDURE p3 (:x INT, :y SMALLINT );
        BEGIN
        INSERT INTO s (snum) VALUES (:x);
        SELECT STATUS INTO :y FROM s WHERE snum = :x;
        END;
    PROCEDURE p4 (:x CHAR(5), :y CHAR(20) );
        BEGIN
        INSERT INTO s (snum,sname) VALUES (:x, :y);
        SELECT sname INTO :y FROM s WHERE snum = :x;
        END;
    END MODULE;
```

Example 3: Creating a stored module containing a stored routines

```
SQL> CREATE MODULE utility_functions
cont>   LANGUAGE SQL
cont> --
cont> -- Define a stored function.
cont> --
cont>   FUNCTION abs (IN :arg INTEGER) RETURNS INTEGER
cont>   COMMENT 'Returns the absolute value of an integer';
cont>      BEGIN
cont>         RETURN CASE
cont>                WHEN :arg < 0 THEN - :arg
cont>                ELSE :arg
cont>         END;
cont>      END;
cont> --
cont> -- Define a stored procedure.
cont> --
cont>   PROCEDURE trace_date (:dt DATE);
cont>      BEGIN
cont>         TRACE :dt;
cont>      END;
cont> --
cont>   FUNCTION mdy (IN :dt DATE) RETURNS CHAR(10)
cont>   COMMENT 'Returns the date in month/day/year format';
cont>      BEGIN
cont>         IF :dt IS NULL THEN
cont>            RETURN '**/**/****';
cont>         ELSE
cont>            CALL trace_date (:dt);
cont>            RETURN CAST(EXTRACT(MONTH FROM :dt) AS VARCHAR(2)) || '/' ||
cont>                   CAST(EXTRACT(DAY FROM :dt) AS VARCHAR(2)) || '/' ||
cont>                  CAST(EXTRACT(YEAR FROM :dt) AS VARCHAR(4));
cont>         END IF;
cont>      END;
cont> END MODULE;
```

Example 4: Using a stored function in a SELECT statement

```
SQL> SELECT mdy(job_end), job_end
cont>   FROM job_history WHERE employee_id = '00164';
             JOB_END
**/**/****   NULL
9/20/1981    20-Sep-1981
2 rows selected
```

Example 5: Using declared local temporary tables in stored procedures

```
SQL> -- The following table must exist in order to execute the following
SQL> -- queries.
SQL> --
SQL> CREATE TABLE payroll
cont> (employee_id CHAR(5),
cont> hours_worked INTEGER,
cont> hourly_sal REAL,
cont> week_date CHAR(10));
SQL> COMMIT;
SQL> --
SQL> -- Create the module containing a declared local temporary table.
SQL> --
SQL> CREATE MODULE paycheck_decl_mod
cont>   LANGUAGE SQL
cont>   DECLARE LOCAL TEMPORARY TABLE module.paycheck_decl_tab
cont>          (employee_id ID_DOM,
cont>           last_name CHAR(14) ,
cont>           hours_worked INTEGER,
cont>           hourly_sal INTEGER(2),
cont>           weekly_pay   INTEGER(2))
cont>           ON COMMIT PRESERVE ROWS
cont> --
cont> -- Create the procedure to insert rows.
cont> --
cont>   PROCEDURE paycheck_ins_decl;
cont>   BEGIN
cont>     INSERT INTO module.paycheck_decl_tab
cont>          (employee_id, last_name, hours_worked, hourly_sal, weekly_pay)
cont>           SELECT p.employee_id, e.last_name,
cont>                    p.hours_worked, p.hourly_sal,
cont>                    p.hours_worked * p.hourly_sal
cont>                 FROM employees e, payroll p
cont>                 WHERE e.employee_id = p.employee_id
cont>                 AND p.week_date = '1995-08-01';
cont>   END;
cont> --
cont> -- Create the procedure to count the low hours.
cont> --
cont>   PROCEDURE low_hours_decl (:cnt INTEGER);
cont>   BEGIN
cont>     SELECT COUNT(*) INTO :cnt FROM module.paycheck_decl_tab
cont>            WHERE hours_worked < 40;
cont>   END;
cont> END MODULE;
SQL> --
SQL> -- Call the procedure to insert the rows.
SQL> --
SQL> CALL paycheck_ins_decl();
SQL> --
SQL> -- Declare a variable and call the procedure to count records with
SQL> -- low hours.
SQL> --
SQL> DECLARE :low_hr_cnt integer;
SQL> CALL low_hours_decl(:low_hr_cnt);
  LOW_HR_CNT
           2

SQL> --
SQL> -- Because the table is a declared local temporary table, you cannot
SQL> -- access it from outside the stored module that contains it.
SQL> --
SQL> SELECT * FROM module.paycheck_decl_tab;
%SQL-F-RELNOTDCL, Table PAYCHECK_DECL_TAB has not been declared in module or
environment
```

Example 6: Creating a stored procedure containing a simple statement

```
SQL> CREATE MODULE a
cont> LANGUAGE SQL
cont> PROCEDURE new_salary_proc
cont> (:id CHAR (5),
cont> :new_salary INTEGER (2));
cont> UPDATE salary_history
cont>              SET salary_end = CURRENT_TIMESTAMP
cont>                  WHERE employee_id = :id;
cont> end module;
```

## CREATE OUTLINE Statement

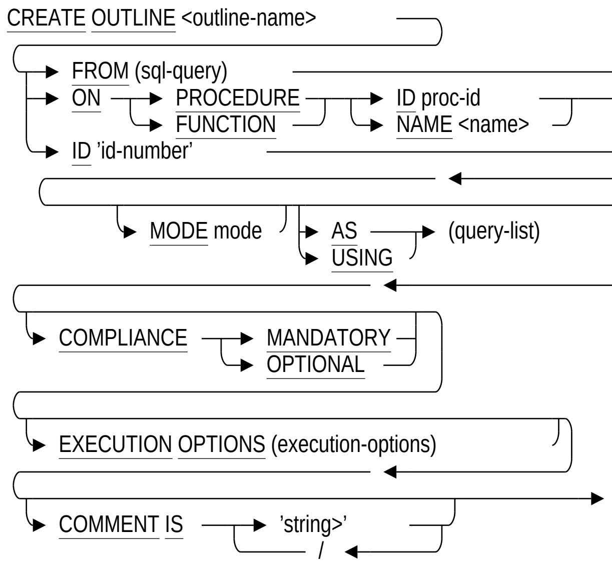
Creates a new query outline and stores this outline in the database.

A query outline is an overall plan for how a query can be implemented and may contain directives that control the join order, join methods, index usage (or all of these) the optimizer selects when processing a query. Use of query outlines helps ensure that query performance is highly stable across releases of Oracle Rdb.

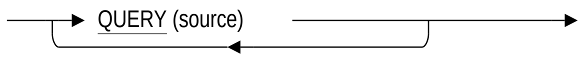
### Environment

You can use the CREATE OUTLINE statement only in interactive SQL.

### Format



query-list =



## CREATE OUTLINE Statement

source =

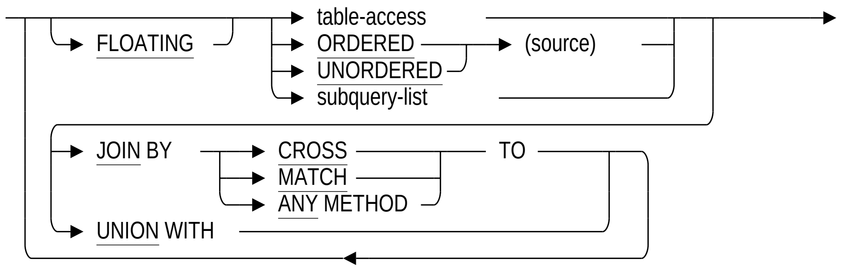
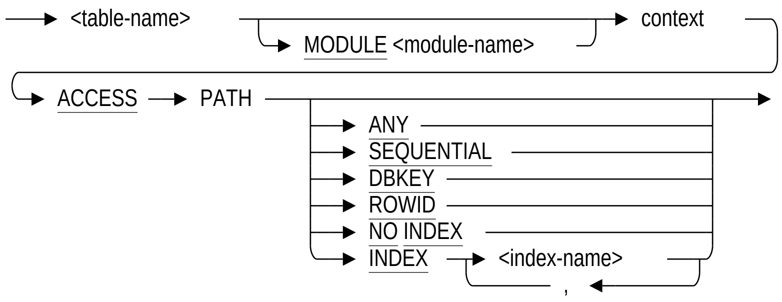
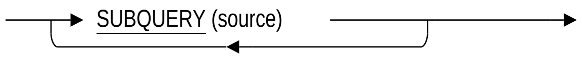


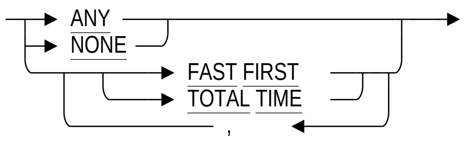
table-access =



subquery-list =



execution-options =



## Arguments

**outline-name**

The name of the new query outline. The name has a maximum length of 31 characters.

**FROM (sql-query)**
Enables an outline to be created directly from an SQL statement.

If the AS clause is not specified, the sql-query is compiled and the resulting outline is stored. If the AS clause is specified, the sql-query provides an alternate means of specifying the ID. If the USING clause is specified, the sql-query is optimized using the designated outline as a starting point.

The only statement accepted as an sql-query in the FROM clause is a SELECT statement. Do not end the sql-query with a semicolon.

**ON PROCEDURE ID proc-id**
**ON PROCEDURE NAME name**
Generates an outline definition for the specified stored procedure that must already exist within the database. During definition of the query outline, the query definitions associated with the specified object are used to check for possible syntax problems, such as referencing a table within the query outline that does not take part within the query of the designated database object. Various exceptions are displayed informing the user of the syntax error.

**ON FUNCTION ID proc-id**
**ON FUNCTION NAME name**
Generates an outline definition for the specified stored function that must already exist within the database. During definition of the query outline, the query definitions associated with the specified object are used to check for possible syntax problems, such as referencing a table within the query outline that does not take part within the query of the designated database object. Various exceptions are displayed informing the user of the syntax error.

**ID 'id-number'**
Specifies the internal hash identification number of the request to which this outline should be applied. Specify a 32-byte string representing a 32-hexadecimal character identification code. The internal hash identification code is generated by the optimizer whenever query outlines are created by the Oracle Rdb optimizer during optimization.

You can optionally specify the MODE clause. You are required to specify the AS clause. You cannot specify the USING clause with the ID id-number clause.

**MODE mode**
Mode is a value assigned to an outline when it is generated by the optimizer. The default mode is 0. Specify a signed integer.

If you create multiple outlines for a single query, the outlines cannot have the same outline mode. When more than one outline exists for a query, you can set the RDMS$BIND_OUTLINE_MODE logical name or RDB_BIND_OUTLINE_MODE configuration parameter to the value of the outline mode for the outline you want the optimizer to use. For example, if you have a query that runs during the day and at night and you created two outlines for the query, you could keep the default outline mode of 0 for the outline to be used during the day, and assign an outline mode of –1 for the outline to be used at night. By setting the RDMS$BIND_OUTLINE_MODE logical name or the RDB_BIND_OUTLINE_MODE configuration parameter to –1 at night, the appropriate outline is run at the appropriate time.

Valid values for modes are –2,147,483,648 to 2,147,483,647. Positive mode values are reserved for future use, so it is recommended that you specify a value between 0 and –2,147,483,648 for the mode value.

**AS (query-list)**
Provides the main definition of an outline.

This clause is only required when creating an outline using the ID id-number clause.

**USING (query-list)**
Specifies the outline to be used for compilation of the contents of the FROM, ON PROCEDURE, and the ON FUNCTION clauses.

You cannot use this clause with the ID id-number clause.

**QUERY**
Specifies that the data sources within the parentheses belong to a separate query.

**FLOATING**
Specifies that the following data source should be considered to be floating and that the order of the data source relative to the other data sources within the same level is not fixed.

**table-name**
Specifies the name of a database table.

**MODULE module-name**
Associates an outline with a declared local temporary table by qualifying the table name with the name of the stored module. In order to apply the outline to the declared local temporary table, the keyword MODULE is required.

**context**
Specifies the context number for this table. Specify an unsigned integer. This number is allocated to the table by the optimizer during optimization. Context numbers are unique within queries.

**ACCESS PATH ANY**
**ACCESS PATH SEQUENTIAL**
**ACCESS PATH DBKEY**
**ACCESS PATH ROWID**
**ACCESS PATH NO INDEX**
Specifies the access path to use to retrieve data from the underlying database table. The following table lists the valid access paths.

| Path | Meaning |
|---|---|
| ANY | Indicates that the optimizer may choose the most appropriate method. |
| SEQUENTIAL | Indicates that sequential access should be used. |
| DBKEY[1] | Indicates the access by database key should be used. |
| ROWID[1] | Indicates the access by database key should be used. |
| NO INDEX | Indicates that any access path not requiring an index can be used. NOINDEX is accepted as a synonym for NO INDEX. |

[1]DBKEY and ROWID are synonyms to each other.

There is no default access path. An access path must be specified for each database table specified within a query outline definition.

**ACCESS PATH INDEX index-name**
Specifies that data should be retrieved using the specified index or list of indexes. If more that one index can be used, then separate each index name with a comma.

Any index name specified should indicate an existing index associated with the table with which the access method is associated.

**ORDERED**
Specifies that all nonfloating data sources within the parentheses should be retrieved in the order specified. Join items in the group are placed adjacently.

**UNORDERED**
Specifies that all data sources within the parentheses should be considered floating and that no order is implied. Join items in the group are placed adjacently.

**SUBQUERY**
Specifies that the data sources within the parentheses belong to a separate subquery.

**JOIN BY CROSS**
**JOIN BY MATCH**
**JOIN BY ANY METHOD**
Specifies the method with which two data sources should be joined. The following table lists the valid methods.

| Method | Meaning |
|---|---|
| CROSS | Indicates that a cross strategy should be used |
| MATCH | Indicates that a match strategy should be used[1] |
| ANY METHOD | Indicates that the optimizer can choose any method to join the two data sources |

[1]The match join strategy requires that an equivalent join column exist between the inner and outer context of the join order. If the query for which the outline is created does not have an equivalent join column, then the optimizer cannot use the match join strategy specified in the outline.

There is no default join method.

**UNION WITH**
Specifies the union of two data sources.

Either a join or union method must be specified between all data sources with the exception of QUERY source blocks.

**Note**

When a join method appears immediately before an ordered or unordered group, the join method is associated with the first join item named in the group.

The union strategy is only valid for queries that use the UNION operator, and all queries that specify the UNION operator must use the union strategy.

**COMPLIANCE MANDATORY**
**COMPLIANCE OPTIONAL**
Specifies the compliance level for this outline.

MANDATORY indicates that all outline directives such as table order and index usage should be followed as specified. If the optimizer is unable to follow any outline directive, an exception is raised.

OPTIONAL indicates that all outline directives are optional and that if they cannot be followed, no exception should be raised. If OPTIONAL is specified, the strategy chosen by the optimizer to carry out the underlying request may not match the strategy specified within the outline.

Use MANDATORY when the strategy that the optimizer chooses must be followed exactly as specified from version to version of Oracle Rdb even if the optimizer finds a more efficient strategy in a future version of Oracle Rdb.

The default is COMPLIANCE OPTIONAL.

**EXECUTION OPTIONS (execution-options)**
Specifies options that the optimizer should take into account during optimization. The following table lists the valid options.

| Option | Meaning |
|---|---|
| ANY | Indicates that the optimizer can choose any optimization method |
| FAST FIRST | Indicates that the optimizer can use FAST FIRST optimization if and when appropriate |
| NONE | Indicates that optional optimizations should not be applied |
| TOTAL TIME | Indicates that the optimizer can use TOTAL TIME optimization if and when appropriate |

The default is EXECUTION OPTIONS (ANY).

**COMMENT IS ’string’**
Adds a comment about the outline. SQL displays the text when it executes a SHOW OUTLINES statement in interactive SQL. Enclose the comment in single quotation marks (’) and separate multiple lines in a comment with a slash mark (/).

## Usage Notes

- See the chapter on the Query Optimizer in the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on using the optimizer to create an outline and customizing query outlines.
- You must have the CREATE privilege on all tables referenced in the query outline.
- The CREATE OUTLINE statement is an online operation. Other users can be attached to the database when an outline is created.
- Each query outline can only contain one SQL statement.
- You can specify Ss (an uppercase *S* followed by a lowercase *s* enclosed in double quotation marks) with the RDMS$DEBUG_FLAGS logical name or RDB_DEBUG_FLAGS configuration parameter to display outlines generated by the optimizer. Or specify the SET FLAGS 'OUTLINE' statement. See the SET FLAGS Statement and the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information.
- The maximum length for each string literal in a comment is 1024 characters.
- Because indexes cannot be defined on a declared local temporary table, the only table access paths allowed are SEQUENTIAL, DBKEY, or ROWID when using the MODULE clause.
- You cannot use a nonstored module name with the MODULE clause.
- You can only specify stored routines with the ON PROCEDURE or ON FUNCTION clauses. If you specify an external routine, SQL generates an error.

## Examples

Example 1: Creating an outline named AVAILABLE_EMPLOYEES

```
SQL> CREATE OUTLINE available_employees
cont> ID '09ADFE9073AB383CAABC4567BDEF3832'  MODE 0
cont> AS (
cont>    QUERY (
cont> --
cont> -- Cross the employees table with departments table first.
cont> --
cont>          employees 0 ACCESS PATH SEQUENTIAL JOIN BY MATCH TO
cont>          departments 3 ACCESS PATH INDEX dept_index JOIN BY MATCH TO
cont>          SUBQUERY (
cont>              job_fitness 2 ACCESS PATH INDEX job_fit_emp, job_fit_dept
cont>           JOIN BY CROSS TO
cont>              SKILLS 4 ACCESS PATH ANY
cont>                       ) JOIN BY MATCH TO
cont>         SUBQUERY (
cont>              major_proj 1 ACCESS PATH ANY JOIN BY CROSS TO
cont>              education 6 ACCESS PATH ANY
cont>                        ) JOIN BY CROSS TO
cont>         research_projects 5 ACCESS PATH ANY UNION WITH
cont> --
cont> -- Always do the union with employees table last
cont> --
cont>         employees 7 ACCESS PATH ANY
cont>          )
cont>   )
cont> COMPLIANCE OPTIONAL
cont> COMMENT IS 'Available employees';
```

Example 2: Creating an outline using the FROM clause

```
SQL> CREATE OUTLINE degrees_for_emps_over_65
cont> FROM
cont>    (SELECT e.last_name, e.first_name, e.employee_id,
cont>              d.degree, d.year_given
cont>          FROM employees e, degrees d
cont>          WHERE e.birthday < ’31-Dec-1930’
cont>          AND e.employee_id = d.employee_id
cont>          ORDER BY e.last_name)
cont> USING
cont>    (QUERY
cont>       (SUBQUERY
cont>          (degrees 1 ACCESS PATH SEQUENTIAL
cont>           JOIN BY CROSS TO
cont>           employees 0 ACCESS PATH ANY
cont>          )
cont>       )
cont>    )
cont> COMPLIANCE OPTIONAL
cont> COMMENT IS ’Outline to find employees over age 65 with college degrees’;
SQL> --
SQL> SHOW OUTLINE degrees_for_emps_over_65
     DEGREES_FOR_EMPS_OVER_65
 Comment:       Outline to find employees over age 65 with college degrees
 Source:

-- Rdb Generated Outline : 13-NOV-1995 15:28
create outline DEGREES_FOR_EMPS_OVER_65
id ’B6923A6572B28E734D6F9E8E01598CD8’
mode 0
as (
  query (
    subquery (
      DEGREES 1       access path sequential
        join by cross to
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance optional     ;
```

Example 3: Creating an outline using the ON FUNCTION clause

```
SQL> CREATE OUTLINE out1
cont> ON FUNCTION NAME function1;
SQL> COMMIT;
SQL> SHOW OUTLINE out1
     OUT1
 Source:
```

```
-- Rdb Generated Outline :  2-FEB-1996 15:46
create outline OUT1
id '264A6DDADCB483AE5B2CDF629C9C8C0F'
mode 0
as (
  query (
    subquery (
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance optional     ;
```

Example 4: Creating an outline on a procedure that accesses a declared local temporary table (see the CREATE MODULE Statement for the stored procedure and temporary table definition)

```
SQL> CREATE OUTLINE outline1
cont> ON PROCEDURE NAME paycheck_ins_decl
cont> MODE 0
cont> AS (
cont>     QUERY (
cont>            module.paycheck_decl_tab MODULE paycheck_decl_mod
cont>            0
cont>            ACCESS PATH SEQUENTIAL
cont>           )
cont>    )
cont> COMPLIANCE OPTIONAL;
SQL> SHOW OUTLINE outline1
     OUTLINE1
 Source:

create outline OUTLINE1
mode 0
as (
  query (
    PAYCHECK_DECL_TAB       MODULE PAYCHECK_DECL_MOD 0
    access path sequential
    )
  )
compliance optional     ;
```

## CREATE PROCEDURE Statement

Creates an external procedure as a schema object in an Oracle Rdb database.

The CREATE PROCEDURE statement is documented under the Create Routine Statement. For complete information on creating an external procedure definition, see the Create Routine Statement.

## Create Routine Statement

Creates an external routine definition as a schema object in an Oracle Rdb database. **External routine** refers to both external functions and external procedures. A routine definition stores information in the database about a subprogram (a function or procedure) written in a 3GL language. The routine definition and the routine image are independent of each other, meaning one can exist without the other. However, to invoke an external routine, you need both the routine definition and routine image.

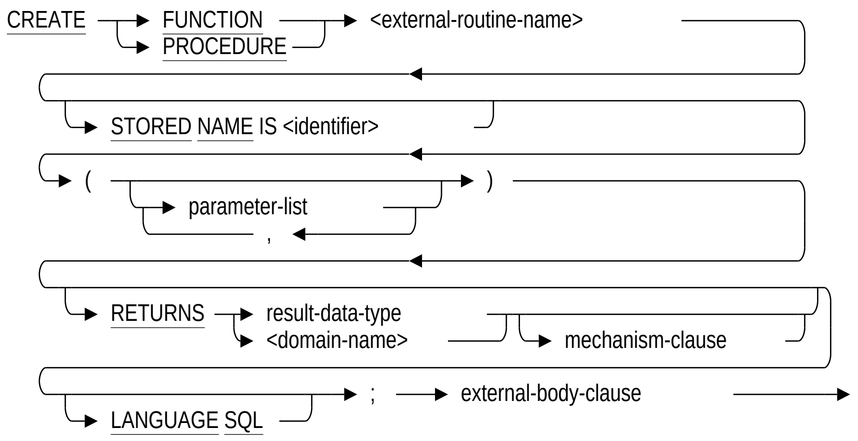
SQL can invoke an external function from anywhere you can specify a value expression. External procedures are invoked using the CALL Statement for Compound Statements.

### Environment

You can use the CREATE FUNCTION and CREATE PROCEDURE statements:

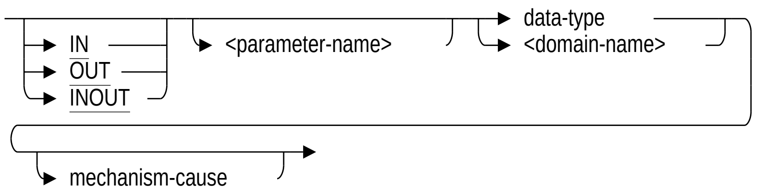
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

### Format

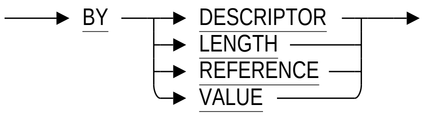


## Create Routine Statement

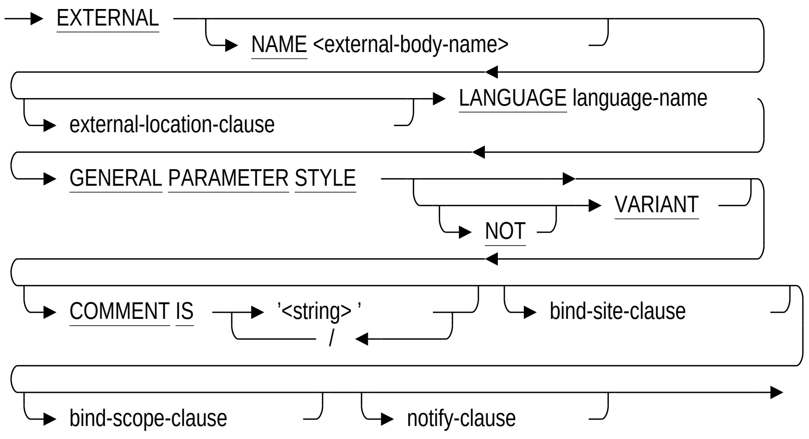
parameter-list =



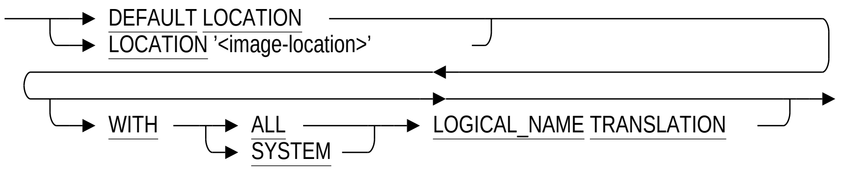
mechanism-clause =



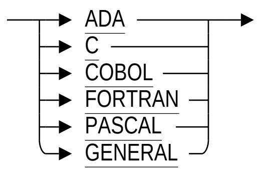
external-body-clause =



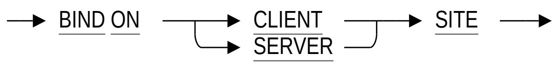
external-location-clause =



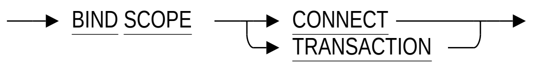
language-name =



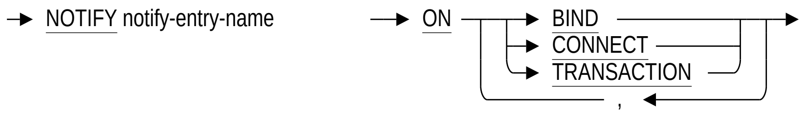
bind-site-clause =



bind-scope-clause =



notify-clause =



## Arguments

**FUNCTION**
Creates an external function definition.

A function optionally accepts a list of IN parameters, always returns a value, and is referenced by name as an element of a value expression.

**PROCEDURE**
Creates an external procedure definition.

A procedure optionally accepts a list of IN, OUT, or INOUT parameters, never returns a value, and is referenced by name in a CALL statement.

**external-routine-name**
The name of the external routine. The name must be unique among external and stored routines in the schema and can be qualified with an alias or, in a multischema database, a schema name.

### STORED NAME IS identifier

The name that Oracle Rdb uses to access the routine when defined in a multischema database. The stored name allows you to access multischema definitions using interfaces, such as RMU, that do not recognize multiple schemas in one database. You cannot specify a stored name for a routine in a database that does not allow multiple schemas. For more information about stored names, see Section 2.2.4.

### parameter-list

The optional parameters of the external routine. For each parameter you can specify a parameter access mode (IN, OUT, and INOUT), a parameter name, a data type, and a passing mechanism (by DESCRIPTOR, LENGTH, REFERENCE, or VALUE).

The parameter access mode (IN, OUT, and INOUT) is optional and specifies how the parameter is accessed (whether it is read, written, or both). IN signifies read only, OUT signifies write only, and INOUT signifies read and write. The parameter access mode defaults to IN.

Only the IN parameter access mode may be specified with parameters to an external function. Any of the parameter access modes (IN, OUT, and INOUT) may be specified with parameters to an external procedure.

The optional parameter name is prefixed with a colon ( : ). The parameter name must be unique within the external routine parameters.

The data type is required and describes the type of parameter using either an SQL data type or a domain name.

You cannot declare a parameter as one of the LIST OF BYTE VARYING, BYTE VARYING, LIST OF VARBTYE, or VARBYTE data types.

### mechanism-clause

Defines the passing mechanism. The following list describes the passing mechanisms.

OpenVMS VAX OpenVMS Alpha

- BY DESCRIPTOR

  Allows passing character data with any parameter access mode to routines compiled by language compilers that implement the OpenVMS calling standard. BY DESCRIPTOR is available only on the OpenVMS platforms. ♦

- BY LENGTH

  A generalized mechanism that allows passing character data with any parameter access mode to routines compiled by language compilers that require reference, and possibly length, information for the parameter.

For OpenVMS, the LENGTH passing mechanism is the same as the DESCRIPTOR passing mechanism. For Digital UNIX, the LENGTH passing mechanism provides the parameter address as the REFERENCE passing mechanism and, for certain languages that require a length value, passes length information to complete the definition of the parameter.

- BY REFERENCE

  Allows passing data with any parameter access mode as a reference to the actual data.

  This is the default passing mechanism for parameters. This is also the default passing mechanism for a function value returning character data.

- BY VALUE

  Allows passing data with the IN parameter access mode to a routine as a value and allows functions to return a value.

  This is the default passing mechanism for a function value returning noncharacter data.

**RETURNS result-data-type**
**RETURNS domain-name**
Describes an external function (returned) value. You can specify a data type and a passing mechanism (BY DESCRIPTOR, LENGTH, REFERENCE, or VALUE). The function value is, by definition, an OUT access mode value.

The data type is required and describes the type of parameter using either an SQL data type or a domain name.

You cannot declare a function value as one of the LIST OF BYTE VARYING, BYTE VARYING, LIST OF VARBYTE, or VARBYTE data types.

**LANGUAGE SQL**
Names the language that calls the routine. The keyword SQL is the only valid response.

**external-body-clause**
Identifies key characteristics of the routine: its name, where the executable image of the routine is located, the language in which the routine is coded, and so forth.

**external-body-name**
The name of the external routine. If you do not specify a name, SQL uses the name you specify in the external-routine-name clause.

This name defines the routine entry address that is called for each invocation of the routine body. The named routine must exist in the external routine image selected by the location clause.

Unquoted names are converted to uppercase characters.

**external-location-clause**
A file specification referencing the image that contains the routine body and optional notify entry points.

**DEFAULT LOCATION**
**LOCATION 'image-location'**
A default or specific location for the external routine image.

On OpenVMS, this can be an image file specification or merely a logical name.

On Digital UNIX, the file specification can represent only a relative or an absolute file definition.

SQL selects a routine based on a combination of factors:

- Image string

  The location defaults to DEFAULT LOCATION, which represents the file specification string RDB$ROUTINES.

OpenVMS VAX OpenVMS Alpha

- Logical name translation

  The WITH ALL LOGICAL_NAME TRANSLATION and the WITH SYSTEM LOGICAL_NAME TRANSLATION clauses specify how logical names in the location string are to be translated.

  If no translation option is specified, or if WITH ALL LOGICAL_NAME TRANSLATION is specified, logical names are translated in the default manner.

  If WITH SYSTEM LOGICAL_NAME TRANSLATION is specified, any logical names in the location string are expanded using only EXECUTIVE_MODE logical names from the SYSTEM logical name table. ♦

**LANGUAGE language-name**
The name of the host language in which the external routine was coded. You can specify ADA, C, COBOL, FORTRAN, PASCAL, or GENERAL. The GENERAL keyword allows you to call routines written in any language.

See the Usage Notes for more language-specific information.

**GENERAL PARAMETER STYLE**
Passes arguments and returns values in a manner similar to the OpenVMS convention for passing arguments and returning function values.

**VARIANT**
**NOT VARIANT**
Controls the evaluation of a function in the scope of a query:

- VARIANT

  Specifying the VARIANT clause forces evaluation of corresponding functions (in the scope of a single query) every time the function appears. If a function could return a different result each time it is invoked, you should use the VARIANT clause.

- NOT VARIANT

  Specifying the NOT VARIANT clause can result in a single evaluation of corresponding function expressions (in the scope of a single query), such that the resulting value is used in all occurrences of the corresponding function expression. When you use the NOT VARIANT clause, Oracle Rdb determines whether or not to invoke the function each time the clause is used.

  For example:

  ```
  SELECT * FROM T1 WHERE F1() > 0 AND F1() < 20;
  ```

  If you define the F1 function as NOT VARIANT, the function F1() may be evaluated just once depending on the optimizer. If you define the F1 function as VARIANT, the function F1() is evaluated twice.

Use of this clause with an external procedure generates an exception.

NOT VARIANT is the default.

**COMMENT IS 'string'**
A description about the nature of the external routine. SQL displays the text of the comment when you execute a SHOW FUNCTION (DESCRIPTION), SHOW PROCEDURE (DESCRIPTION), or a SHOW ROUTINE (DESCRIPTION) statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash (/).

**BIND ON CLIENT SITE**
**BIND ON SERVER SITE**
Selects the execution model and environment for external routine execution.

OpenVMS VAX OpenVMS Alpha

CLIENT SITE binding is available, and is the default, only on OpenVMS platforms. When you specify CLIENT SITE binding, the external routine is activated in the same process as the Oracle Rdb server, which may also be the client application or SQL process. ♦

SERVER SITE binding is available on all platforms and causes the external routine to be activated in a separate process that is on the same processing node as the Oracle Rdb server.

**BIND SCOPE CONNECT**
**BIND SCOPE TRANSACTION**

Defines the scope during which an external routine is activated and at what point the external routine is deactivated. The default scope is CONNECT.

- CONNECT

  An active routine is deactivated when you detach from the database (or exit without detaching).

- TRANSACTION

  An active routine is deactivated when a transaction is terminated (COMMIT or ROLLBACK). In the event that a transaction never occurs, the scope reverts to CONNECT.

**notify-clause**

Specifies the name of a second external routine called (notified) when certain external routine or database-related events occur. This name defines the routine entry address that is called, for each invocation of the notify routine. The named routine must exist in the external routine image selected by the location clause.

Unquoted names are converted to uppercase characters.

The events of interest to the notify routine are ON BIND, ON CONNECT, and ON TRANSACTION. Multiple events can be specified.

The following describes the events and scope of each event:

| | |
|---|---|
| BIND | Routine activation to routine deactivation |
| CONNECT | Database attach to database disconnect |
| TRANSACTION | Start transaction to commit or roll back transaction |

## Usage Notes

- You can invoke an external function from any SQL value-expression argument, such as in a WHERE clause, SELECT statement, INSERT statement, COMPUTED BY clause, stored procedure, constraint definitions, or trigger definitions.

  For information about invoking an external function from triggers, see the *Oracle Rdb7 Guide to Database Design and Definition*.
- You can invoke an external procedure using the CALL statement within a compound statement.
- Certain combinations of parameter data type and passing mechanisms may not be available on all platforms or for all languages. For example, BIGINT . . . BY VALUE is not available on the OpenVMS VAX platform.
- No more than 255 arguments may be passed to an external routine. For Digital UNIX FORTRAN, passing a character string using the BY LENGTH mechanism generates two arguments for a single character string parameter.
- Certain languages do not accept parameters passed by VALUE.
- Certain languages, such as FORTRAN, are defined by syntax that prevents returning function values by REFERENCE.
- The procedure parameter access modes, INOUT and OUT, are incompatible with the BY VALUE passing mechanism for external procedures.
- Only minimal defaults are provided for the location file specification on OpenVMS. If not provided as part of the file specification or logical name, the device and directory default to the current default device and directory (SYS$DISK:[] if SYS$DISK references only a device or devices; or SYS$DISK: if SYS$DISK is a search list that references a directory). There is no default for file type.
- No defaults are provided for the location file specification on Digital UNIX. There is no metacharacter translation to support shell-like variable value substitution or file name substitution. The logical name translation options are not applicable.
- The language used to implement the external routine may limit the data types that can be specified. Refer to the language-specific documentation for more information.

- Specifying a specific language can alter the passing mechanism semantics for parameters and function values with character data types. Language C causes character data types passed by REFERENCE to be passed as null-terminated strings. On Digital UNIX, PASCAL and FORTRAN cause character data types passed by LENGTH to pass both a reference to the data and a length value. The GENERAL keyword can be used instead of the actual host language name to bypass any of these language-specific passing mechanism semantics.
- The GENERAL PARAMETER STYLE clause does not allow arguments that have NULL values.
- The maximum length for each string literal in a comment is 1024 characters.
- An external routine can attach to databases and execute SQL data manipulation statements using those databases, for example, through embedded SQL.
- An external routine cannot execute data definition statements.
- A single routine image may be referenced by multiple routines defined in a single database or by routines registered in multiple, attached databases.
- See the *Oracle Rdb7 Guide to SQL Programming* for more information about:
  - Creating external routines
  - Invoking external routines from applications
  - Parameters and passing mechanisms
  - Routine activation and deactivation
  - Using notification routines
  - Execution environments
  - Exceptions
  - Limitations
  - Recommendations
  - Common problems and solutions
- You must execute this statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with the characteristics specified in the most recent DECLARE TRANSACTION statement.

- You cannot execute this statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on the RDB$SYSTEM storage area.

## Examples

Example 1: System provided integer absolute value routine

OpenVMS VAX

On OpenVMS VAX:

```
SQL> CREATE FUNCTION IABS (IN INTEGER BY REFERENCE)
cont>   RETURNS INTEGER BY VALUE;
cont>   EXTERNAL NAME MTH$JIABS
cont>   LOCATION 'SYS$SHARE:MTHRTL.EXE'
cont>   LANGUAGE GENERAL
cont>   GENERAL PARAMETER STYLE
cont>   VARIANT;
SQL> --
SQL> SELECT IABS(-33) FROM JOBS LIMIT TO 1 ROW;

          33
1 row selected
```

♦

OpenVMS Alpha

On OpenVMS Alpha:

```
SQL> CREATE FUNCTION IABS (IN INTEGER BY REFERENCE)
cont>   RETURNS INTEGER BY VALUE;
cont>   EXTERNAL NAME MTH$JIABS
cont>   LOCATION 'SYS$SHARE:DPML$SHR.EXE'
cont>   LANGUAGE GENERAL
cont>   GENERAL PARAMETER STYLE
cont>   VARIANT;
SQL> --
SQL> SELECT IABS(-33) FROM JOBS LIMIT TO 1 ROW;

          33
1 row selected
```

♦

Digital UNIX

On Digital UNIX:

```
SQL> SET QUOTING RULES 'SQL92';
SQL> CREATE FUNCTION iabs (IN INTEGER BY REFERENCE)
cont>   RETURNS INTEGER BY VALUE;
cont>   EXTERNAL NAME "r_jabs"
cont>   LOCATION '/usr/shlib/libm.so'
cont>   LANGUAGE GENERAL
cont>   GENERAL PARAMETER STYLE
cont>   VARIANT
cont>   BIND ON SERVER SITE;
SQL> --
SQL> SELECT iabs(-33) FROM jobs LIMIT TO 1 ROW;

          33
1 row selected
```

♦

Example 2: Using the VARIANT clause, instead of the NOT VARIANT clause

OpenVMS VAX OpenVMS Alpha

The first CREATE FUNCTION statement in the following example creates a function with the NOT VARIANT clause. The NOT VARIANT clause indicates to Oracle Rdb that the function would return the same result no matter how many times it is called. Because the argument is a string literal (and could never change), Oracle Rdb optimizes the entire function call so that it is not called in subsequent select statements.

```
SQL> -- Create a function with a NOT VARIANT clause.
SQL> CREATE function DO_COM (IN VARCHAR(255) BY DESCRIPTOR)
cont>       RETURNS INTEGER;
cont>       EXTERNAL NAME LIB$SPAWN
cont>                LOCATION 'SYS$SHARE:LIBRTL.EXE'
cont>       LANGUAGE GENERAL
cont>       GENERAL PARAMETER STYLE
cont>       NOT VARIANT;
SQL> --
SQL> -- Use a SELECT statement to pass a string literal to the function.
SQL> --
SQL> -- Because Oracle Rdb optimizes functions with the NOT VARIANT
SQL> -- clause, and the function is passed a string literal,
SQL> -- Oracle Rdb does not call the function from subsequent
SQL> -- statements.
SQL> --
SQL> SELECT DO_COM('WRITE SYS$OUTPUT "HELLO"'), employee_id FROM employees
cont> LIMIT TO 5 ROWS;
HELLO
      DO_COM   EMPLOYEE_ID
           1   00164
           1   00165
           1   00166
           1   00167
           1   00168
5 rows selected
SQL> --
SQL> -- Use the VARIANT clause to create the function:
SQL> --
SQL> CREATE function DO_COM (IN VARCHAR(255) BY DESCRIPTOR)
cont>       RETURNS INTEGER;
cont>       EXTERNAL NAME lib$SPAWN
cont>                LOCATION 'SYS$SHARE:LIBRTL.EXE'
cont>       LANGUAGE GENERAL
cont>       GENERAL PARAMETER STYLE
cont>       VARIANT;
SQL> SELECT DO_COM('WRITE SYS$OUTPUT "HELLO"'), EMPLOYEE_ID FROM EMPLOYEES
cont> LIMIT TO 5 ROWS;
HELLO
HELLO
      DO_COM   EMPLOYEE_ID
           1   00164
HELLO
           1   00165
HELLO
           1   00166
HELLO
           1   00167
           1   00168
5 rows selected
```

♦

Example 3: User-defined external function definition

Digital UNIX

On Digital UNIX:

```
$ cat mod.c
extern int MOD( int *dividend, int *divisor )
{return (*dividend - ((*dividend / *divisor) * (*divisor)));}
$
$ cc -c -o mod.o mod.c
$ ld -shared -o mod.so mod.o -lots -lc
$ sql
$
SQL> attach 'file personnel';
SQL> create function mod (integer, integer) returns integer;
cont>    external location 'mod.so'
cont>    language c
cont>    general parameter style
cont>    bind on server site;
SQL> show function mod
Function name is: MOD
 Function ID is: 1
 External Location is: mod.so
 Bind on Server Site
 Bind Scope Connect
 Language is: C
 GENERAL parameter passing style used
 Number of parameters is: 2
                                Data Type
                                ---------

                                INTEGER
        Function result datatype
        Return value is passed by VALUE

                                INTEGER
        Parameter position is 1
        Parameter is IN (read)
        Parameter is passed by REFERENCE

                                INTEGER
        Parameter position is 2
        Parameter is IN (read)
        Parameter is passed by REFERENCE

SQL> select mod (123, 17) from jobs limit to 1 row;

           4
1 row selected
```

♦

Example 4: External function and external procedure definition

The following OpenVMS example demonstrates:

OpenVMS VAX OpenVMS Alpha

- An external function and an external procedure
- Both CLIENT SITE and SERVER SITE binding
- BIND SCOPE
- A NOTIFY routine and events
- SQL callback using embedded SQL and SQL module language
- SQL$PRE options required when using callback
- Linker options required when using callback

In this example, a new column is added to the EMPLOYEES table in the MF_PERSONNEL database. External routines are used to set this column to spaces and to the SOUNDEX value corresponding to the various employee names. Transaction control at the application level (in this instance, in SQL) in conjunction with a notify routine demonstrates how the actions of the external routines can be affected by actions of the application.

The space-filling is performed by an external function, CLEAR_SOUNDEX, (written in C) containing embedded SQL, which opens another instance of the MF_PERSONNEL database and leaves it open until deactivated.

The SOUNDEX name-setting is performed by an external procedure (written in FORTRAN), assisted by a notify routine (written in FORTRAN) which performs the database connection and transaction control. All the database operations are performed by SQL module language routines. The procedure also opens another instance of the MF_PERSONNEL database, which is disconnected by the notify routine when the routine is deactivated at the end of the transaction. Display statements executed by the notify routine serve to demonstrate the progress of the database operations.

```
SQL> ATTACH ’FILENAME MF_PERSONNEL’;
SQL> --
SQL> -- Add the new column SOUNDEX_NAME to the EMPLOYEES table.
SQL> --
SQL> ALTER TABLE EMPLOYEES ADD SOUNDEX_NAME CHAR(4);
SQL> --
SQL> -- Define the CLEAR_SOUNDEX function.
SQL> --
SQL> CREATE FUNCTION CLEAR_SOUNDEX ()
cont>  RETURNS INTEGER BY VALUE;
cont>     EXTERNAL NAME CLEAR_SOUNDEX
cont>       LOCATION ’CLEAR_SOUNDEX.EXE’
cont>     LANGUAGE C GENERAL PARAMETER STYLE VARIANT
cont>     BIND ON SERVER SITE BIND SCOPE CONNECT;
SQL> --
SQL> -- Define the ADD_SOUNDEX_NAME procedure.
SQL> --
SQL> CREATE PROCEDURE ADD_SOUNDEX_NAME
cont>  (INOUT INTEGER BY REFERENCE);
cont>     EXTERNAL NAME ADD_SOUNDEX_NAME
cont>       LOCATION ’ADD_SOUNDEX.EXE’
cont>     LANGUAGE FORTRAN GENERAL PARAMETER STYLE
cont>     BIND ON CLIENT SITE BIND SCOPE TRANSACTION
cont>     NOTIFY ADD_SOUNDEX_NOTIFY ON BIND, TRANSACTION;
SQL> --
SQL> COMMIT;
SQL> DISCONNECT ALL;
SQL> EXIT;
```

Example 5: The CLEAR_SOUNDEX.SC program written in C

```
/* Set the soundex_name column to spaces, return any error as function value */

static int state = 0;

extern int clear_soundex () {
    exec sql include sqlca ;
    exec sql declare alias filename MF_PERSONNEL;
    if (state == 0) {
 exec sql attach 'filename MF_PERSONNEL';
 state = 1;
    }
    exec sql set transaction read write;
    if (SQLCA.SQLCODE < 0)
 return SQLCA.SQLCODE;
    exec sql update employees set soundex_name = '    ';
    if (SQLCA.SQLCODE < 0)
 return SQLCA.SQLCODE;
    exec sql commit;
    if (SQLCA.SQLCODE < 0)
 return SQLCA.SQLCODE;
    return 0;
}
```

Example 6: Compiling, creating a linker options file, and linking the CLEAR_SOUNDEX program

```
$ SQL$PRE/CC/NOLIST/SQLOPT=ROLLBACK_ON_EXIT CLEAR_SOUNDEX.SC

$ CREATE CLEAR_SOUNDEX.OPT
    UNIVERSAL = CLEAR_SOUNDEX
    PSECT_ATTR=RDB$MESSAGE_VECTOR,NOSHR
    PSECT_ATTR=RDB$DBHANDLE,NOSHR
    PSECT_ATTR=RDB$TRANSACTION_HANDLE,NOSHR

$ LINK/SHARE=CLEAR_SOUNDEX.EXE -
  CLEAR_SOUNDEX.OBJ, SQL$USER:/LIBRARY, -
  CLEAR_SOUNDEX.OPT/OPT
    UNIVERSAL = CLEAR_SOUNDEX
    PSECT_ATTR=RDB$MESSAGE_VECTOR,NOSHR
    PSECT_ATTR=RDB$DBHANDLE,NOSHR
    PSECT_ATTR=RDB$TRANSACTION_HANDLE,NOSHR
```

Example 7: The ADD_SOUNDEX.FOR program written in FORTRAN

```
C  Set the soundex values, returning any error in the IN/OUT parameter

 SUBROUTINE ADD_SOUNDEX_NAME (ERROR)
 CHARACTER ID*5,LAST*14,SX_NAME*4
 INTEGER ERROR
 ERROR = 0
 ID = '00000'
10 CALL GET_NAME (ID, LAST, ERROR)
 IF (ERROR .NE. 0) GO TO 80
 CALL MAKE_SOUNDEX_NAME (LAST, SX_NAME)
 CALL SET_SOUNDEX_NAME (ID, SX_NAME, ERROR)
 IF (ERROR .EQ. 0) GO TO 10
80 IF (ERROR .EQ. 100) ERROR = 0
90 RETURN
 END

C  Perform database connection and transaction operations for notify events

 SUBROUTINE ADD_SOUNDEX_NOTIFY (FUNC, RSV1, RSV2, RSV3)
  INTEGER FUNC, RSV1, RSV2, RSV3, SQLCODE
 SQLCODE = 0
 GO TO (10,20,5,5,30,40,50),FUNC
5 TYPE *,'*** ADD_SOUNDEX_NOTIFY bad func ***'
 GO TO 90
10 TYPE *,'*** ADD_SOUNDEX_NOTIFY activate ***'
 CALL ATTACH_DB (SQLCODE)
 IF (SQLCODE .NE. 0) GO TO 80
 GO TO 90
20 TYPE *,'*** ADD_SOUNDEX_NOTIFY deactivate ***'
 CALL DETACH_DB (SQLCODE)
 IF (SQLCODE .NE. 0) GO TO 80
 GO TO 90
30 TYPE *,'*** ADD_SOUNDEX_NOTIFY start tran ***'
 CALL START_TRAN (SQLCODE)
 IF (SQLCODE .NE. 0) GO TO 80
 GO TO 90
40 TYPE *,'*** ADD_SOUNDEX_NOTIFY commit tran ***'
 CALL COMMIT_TRAN (SQLCODE)
 IF (SQLCODE .NE. 0) GO TO 80
 GO TO 90
50 TYPE *,'*** ADD_SOUNDEX_NOTIFY rollback tran ***'
 CALL ROLLBACK_TRAN (SQLCODE)
 IF (SQLCODE .NE. 0) GO TO 80
 GO TO 90
80 CALL SQL_SIGNAL ()
90 RETURN
 END

C       A 'substitute' SOUNDEX routine for demonstration purposes only
```

```
 SUBROUTINE MAKE_SOUNDEX_NAME (NAME, SOUNDEX_NAME)
 CHARACTER NAME*(*),SOUNDEX_NAME*4
 SOUNDEX_NAME(1:1)=NAME(1:1)
        IV = ICHAR(NAME(1:1))+22
 SOUNDEX_NAME(2:2)=CHAR(MOD(IV,10)+48)
 SOUNDEX_NAME(3:3)=CHAR(MOD(IV/10,10)+48)
 SOUNDEX_NAME(4:4)=CHAR(IV/100+48)
 RETURN
 END
```

Example 8: The ADD_SOUNDEXM.SQLMOD module

```
-- Support for set soundex routine

MODULE ADD_SOUNDEX
LANGUAGE FORTRAN
PARAMETER COLONS

PROCEDURE ATTACH_DB (SQLCODE);
    ATTACH 'FILENAME MF_PERSONNEL';
PROCEDURE DETACH_DB (SQLCODE);
    DISCONNECT DEFAULT;

PROCEDURE START_TRAN (SQLCODE);
    SET TRANSACTION READ WRITE;
PROCEDURE COMMIT_TRAN (SQLCODE);
    COMMIT;
PROCEDURE ROLLBACK_TRAN (SQLCODE);
    ROLLBACK;

PROCEDURE GET_NAME (:ID CHAR(5), :LASTNAME CHAR(14), SQLCODE);
    SELECT EMPLOYEE_ID, LAST_NAME INTO :ID, :LASTNAME
    FROM EMPLOYEES WHERE EMPLOYEE_ID > :ID LIMIT TO 1 ROW;

PROCEDURE SET_SOUNDEX_NAME (:ID CHAR(5), :SX_NAME CHAR(4), SQLCODE);
    UPDATE EMPLOYEES SET SOUNDEX_NAME = :SX_NAME WHERE EMPLOYEE_ID = :ID;
```

Example 9: Compiling, creating the linker options file, and linking the FORTRAN and SQL module language programs

```
$ FORTRAN/NOLIST ADD_SOUNDEX.FOR
$ SQL$MOD ADD_SOUNDEXM.SQLMOD

$ CREATE ADD_SOUNDEX.OPT
    UNIVERSAL = ADD_SOUNDEX_NAME
    UNIVERSAL = ADD_SOUNDEX_NOTIFY
    PSECT_ATTR=RDB$MESSAGE_VECTOR,NOSHR
    PSECT_ATTR=RDB$DBHANDLE,NOSHR
    PSECT_ATTR=RDB$TRANSACTION_HANDLE,NOSHR
```

```
$ LINK/SHARE=ADD_SOUNDEX.EXE -
  ADD_SOUNDEX.OBJ, ADD_SOUNDEXM.OBJ, SQL$USER:/LIBRARY, -
  ADD_SOUNDEX.OPT/OPT
    UNIVERSAL = ADD_SOUNDEX_NAME
    UNIVERSAL = ADD_SOUNDEX_NOTIFY
    PSECT_ATTR=RDB$MESSAGE_VECTOR,NOSHR
    PSECT_ATTR=RDB$DBHANDLE,NOSHR
    PSECT_ATTR=RDB$TRANSACTION_HANDLE,NOSHR
```

Example 10: Using the routines with interactive SQL

```
$ SQL
SQL> ATTACH 'FILENAME MF_PERSONNEL';
SQL> --
SQL> DECLARE :ERROR INTEGER;
SQL> --
SQL> SELECT EMPLOYEE_ID,SOUNDEX_NAME FROM EMPLOYEES
cont>  LIMIT TO 3 ROWS;
 EMPLOYEE_ID   SOUNDEX_NAME
 00165         NULL
 00190         NULL
 00187         NULL
3 rows selected
SQL> COMMIT;
SQL> --
SQL> BEGIN
cont>  SET :ERROR = CLEAR_SOUNDEX ();
cont> END;
SQL> PRINT :ERROR;
       ERROR
           0
SQL> --
SQL> SELECT EMPLOYEE_ID,SOUNDEX_NAME FROM EMPLOYEES
cont>  LIMIT TO 3 ROWS;
 EMPLOYEE_ID   SOUNDEX_NAME
 00165
 00190
 00187
3 rows selected
SQL> COMMIT;
SQL> --
SQL> SET TRANSACTION READ ONLY;
SQl> BEGIN
cont>  SET :ERROR = 0;
cont>  CALL ADD_SOUNDEX_NAME (:ERROR);
cont> END;
*** ADD_SOUNDEX_NOTIFY activate ***
*** ADD_SOUNDEX_NOTIFY start tran ***
SQL> PRINT :ERROR;
       ERROR
           0
SQL> COMMIT;
```

```
*** ADD_SOUNDEX_NOTIFY commit tran ***
*** ADD_SOUNDEX_NOTIFY deactivate ***
SQL> --
SQL> SELECT EMPLOYEE_ID,SOUNDEX_NAME FROM EMPLOYEES
cont>  LIMIT TO 3 ROWS;
 EMPLOYEE_ID   SOUNDEX_NAME
 00165         S501
 00190         O101
 00187         L890
3 rows selected
SQL> COMMIT;
SQL> --
SQL> BEGIN
cont>  SET :ERROR = CLEAR_SOUNDEX ();
cont> END;
SQL> PRINT :ERROR;
       ERROR
           0
SQL> --
SQL> SET TRANSACTION READ ONLY;
SQL> BEGIN
cont>  SET :ERROR = 0;
cont>  CALL ADD_SOUNDEX_NAME (:ERROR);
cont> END;
*** ADD_SOUNDEX_NOTIFY activate ***
*** ADD_SOUNDEX_NOTIFY start tran ***
SQL> PRINT :ERROR;
       ERROR
           0
SQL> ROLLBACK;
*** ADD_SOUNDEX_NOTIFY rollback tran ***
*** ADD_SOUNDEX_NOTIFY deactivate ***
SQL> --
SQL> SELECT EMPLOYEE_ID,SOUNDEX_NAME FROM EMPLOYEES
cont>  LIMIT TO 3 ROWS;
 EMPLOYEE_ID   SOUNDEX_NAME
 00165
 00190
 00187
3 rows selected
SQL> COMMIT;
```
♦

## CREATE SCHEMA Statement

Creates a new schema in the current default catalog of a multischema database.

**Note**

Use of the CREATE SCHEMA statement to create a database is a deprecated feature. If you specify physical attributes of a database such as the root file parameters, you receive an error message, but SQL creates the database anyway.

```
SQL> CREATE SCHEMA FILENAME TEST SNAPSHOT IS DISABLED;
%SQL-I-DEPR_FEATURE, Deprecated Feature: SCHEMA (meaning DATABASE)
```

If you do not specify any physical attributes, you receive an error message noting that you must enable multischema naming.

```
SQL> CREATE SCHEMA PARTS
%SQL-F-SCHCATMULTI, Schemas and catalogs may only be referenced with
multischema enabled
```

A **schema** is a group of definitions within a database. The CREATE SCHEMA statement lets you specify in a single SQL statement all data and privilege definitions for a new schema. You can also add definitions to the schema later.

A **database**, in addition to schema definitions, includes database system files and user data. If you need to specify any physical database characteristics such as the database root file or storage area parameters, use the CREATE DATABASE statement. See the CREATE DATABASE Statement for more information.

You can specify any number of optional schema elements to the CREATE SCHEMA statement. **Schema elements** are any of the CREATE statements (except CREATE STORAGE AREA, CREATE DOMAIN . . . FROM path-name, and CREATE TABLE . . . FROM path-name) or a GRANT statement.

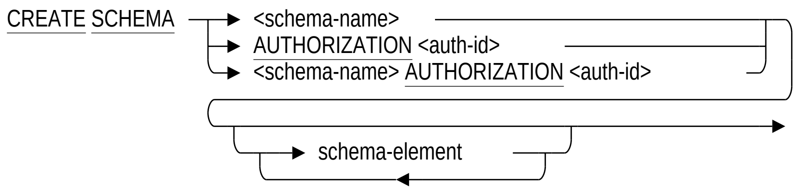
These statements require statement terminators, except when they are part of a CREATE SCHEMA or CREATE DATABASE statement. When you use these statements within a CREATE SCHEMA statement, use a statement terminator on the last schema element only. The first statement terminator that SQL encounters ends the CREATE SCHEMA statement. Later CREATE or GRANT statements are not within the scope of the CREATE SCHEMA statement.

## Environment

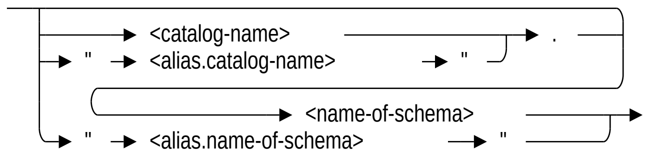
You can use the CREATE SCHEMA statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

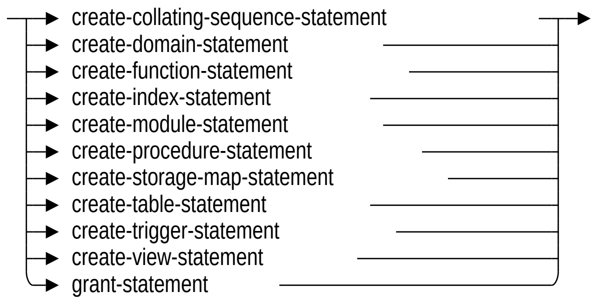
## Format



schema-name =



schema-element =



## Arguments

**schema-name**
Specifies the name of the schema created by the CREATE SCHEMA statement.

You can qualify the schema name with either a catalog name or the catalog name qualified by the alias. You must enclose the alias and catalog name in double quotation marks and separate them with a period. You must issue the SET QUOTING RULES statement before you specify the alias and catalog name pair, or SQL issues an error message about the use of double quotation marks.

For information on qualifying schema names with aliases and catalog names, see Section 2.2.8.

**AUTHORIZATION auth-id**
If you do not specify a schema name, the authorization identifier specifies the default schema.

If you want to comply with the ANSI/ISO 1989 standard, specify the AUTHORIZATION clause without the schema name. Specify both the AUTHORIZATION clause and the schema name to comply with the ANSI/ISO SQL standard.

**schema-element**
Some CREATE statements or a GRANT statement. See the syntax diagram in this section for the complete list of allowable CREATE statements.

OpenVMS VAX OpenVMS Alpha

**create-collating-sequence-statement**
See the CREATE COLLATING SEQUENCE Statement for details.

If you want to specify a collating sequence in a CREATE DOMAIN statement embedded in a CREATE SCHEMA statement, you must first specify a CREATE COLLATING SEQUENCE statement within the same CREATE SCHEMA statement. ♦

**create-domain-statement**
See the CREATE DOMAIN Statement for details.

You cannot use the FROM path-name clause when embedding a CREATE DOMAIN statement in a CREATE SCHEMA statement. You can, however, issue a separate CREATE DOMAIN statement following the CREATE SCHEMA statement. You can also describe the domain directly within the CREATE SCHEMA statement.

If you want to specify a collating sequence in your embedded CREATE DOMAIN statement, you must first specify a CREATE COLLATING SEQUENCE statement within the same CREATE SCHEMA statement.

**create-function-statement**
See the Create Routine Statement for details.

**create-index-statement**
See the CREATE INDEX Statement for details.

**create-module-statement**
See the CREATE MODULE Statement for details.

**create-procedure-statement**
See the Create Routine Statement for details.

**create-storage-map-statement**
See the CREATE STORAGE MAP Statement for details.

**create-table-statement**
See the CREATE TABLE Statement for details.

You cannot use the FROM path-name clause when embedding a CREATE TABLE statement in a CREATE SCHEMA statement. You can, however, issue a separate CREATE TABLE statement following the CREATE SCHEMA statement. You can also describe the table directly within the CREATE SCHEMA statement.

The CREATE TABLE statements in a CREATE SCHEMA statement can refer to domains not yet created, provided that CREATE DOMAIN statements for the domains are in the same CREATE SCHEMA statement.

**create-trigger-statement**
See the CREATE TRIGGER Statement for details.

**create-view-statement**
See the CREATE VIEW Statement for details.

**grant-statement**
See the GRANT Statement for details.

## Usage Notes

- If the CREATE SCHEMA statement is a subordinate clause within a CREATE DATABASE statement, the alias and catalog names must be the same as the alias and catalog names for the last schema created, or must be the default alias and catalog name.

## Example

Example 1: Creating a schema within a multischema database

The following interactive statements create a database that contains a schema within a catalog. You issue the CREATE SCHEMA statement alone or as part of a CREATE DATABASE statement.

```
SQL> SET DIALECT 'SQL92';
SQL> CREATE DATABASE ALIAS PERS_ALIAS FILENAME personnel MULTISCHEMA IS ON;
SQL> CREATE CATALOG "PERS_ALIAS.ADMIN";
SQL> CREATE SCHEMA "PERS_ALIAS.ADMIN".PAYROLL;
SQL> SHOW SCHEMAS;
Schemas in database PERS_ALIAS
    "PERS_ALIAS.ADMIN".PAYROLL
    "PERS_ALIAS.RDB$CATALOG".RDB$SCHEMA
```

## CREATE STORAGE AREA Clause

**Note**

You cannot issue CREATE STORAGE AREA as an independent statement. It is a clause allowed only as part of a CREATE DATABASE or IMPORT statement.

You can also create a storage area using the ADD STORAGE AREA clause of the ALTER DATABASE statement.

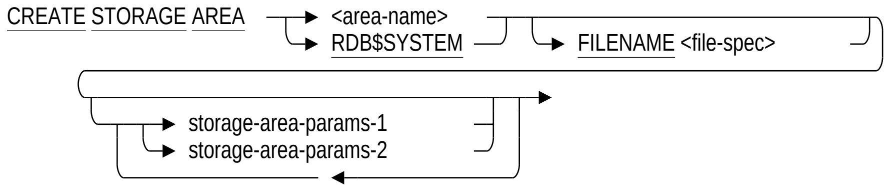
Creates additional storage areas in a multifile database. **Storage areas** are data and snapshot files that are associated with particular tables in a multifile database.

A CREATE STORAGE AREA clause specifies the names for the storage area files and determines their physical characteristics. Subsequent CREATE STORAGE MAP statements associate the storage area with particular tables in the database.

### Environment

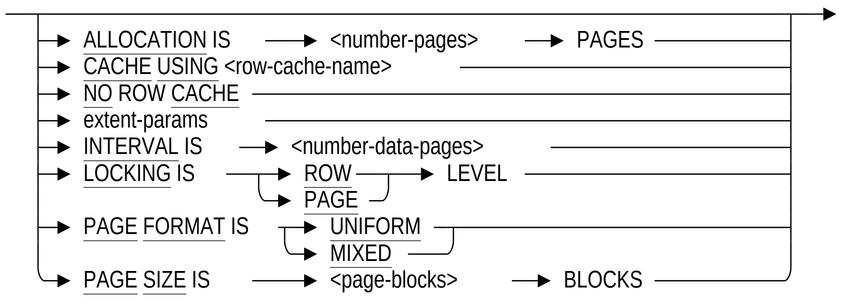
You can use the CREATE STORAGE AREA clause only within a CREATE DATABASE or IMPORT statement.

### Format

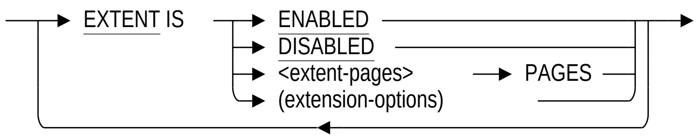


## CREATE STORAGE AREA Clause

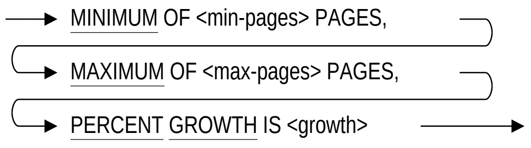
storage-area-params-1 =



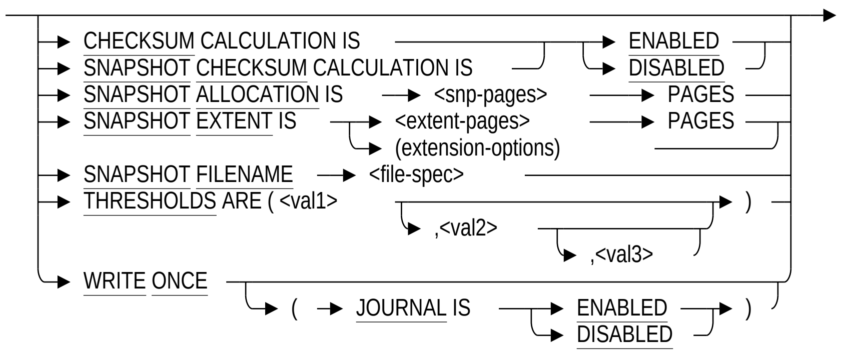
extent-params =



extension-options =


MINIMUM OF <min-pages> PAGES,
MAXIMUM OF <max-pages> PAGES,
PERCENT GROWTH IS <growth>

storage-area-params-2 =



## Arguments

### STORAGE AREA area-name

Specifies the name of the storage area you want to create. The name cannot be the same as any other storage area definition in the database.

If you omit the FILENAME argument, SQL also uses the area name you specify as the file name for storage area files and creates the storage area in the current default directory. These files end with the .rda and .snp extensions.

### STORAGE AREA RDB$SYSTEM

Specifies that you want the CREATE STORAGE AREA clause to override the default characteristics for the main storage area, RDB$SYSTEM, in a new database.

The RDB$SYSTEM storage area contains:

- Database system tables
- Any tables or lists not associated with other storage areas by CREATE STORAGE MAP statements
- Tables explicitly associated with the RDB$SYSTEM storage area by CREATE STORAGE MAP statements

### FILENAME file-spec

Provides an explicit file specification for storage area files. The CREATE STORAGE AREA clause creates two files: a storage area file with a file extension of .rda, and a snapshot file with a file extension of .snp. If you omit the FILENAME argument, the file specification takes the following defaults:

- Device: the current device for the process (on OpenVMS only)
- Directory: the current directory for the process
- File name: the name specified for the storage area

The file specification is used for both the storage area and snapshot files that comprise the storage area (unless you use the SNAPSHOT FILENAME argument to specify a different file for the snapshot file). Because the CREATE STORAGE AREA clause can create two files with different file extensions, do not specify a file extension with the file specification.

Use either a full file specification or a partial file specification.

## CREATE STORAGE AREA Clause

OpenVMS VAX OpenVMS Alpha You can use a logical name for all or part of a file specification. ♦

One benefit of a multifile database is that its files can reside on more than one disk. If you want storage area files to reside on another disk, you must specify the FILENAME argument with a full file specification.

However, you may choose to create a multifile database even if your main purpose in creating the storage area is not to distribute storage area files across more than one disk. For instance, a multifile database enables you to:

- Take advantage of hashed indexes. Hashed indexes require a storage area with mixed page format and cannot be stored in the RDB$SYSTEM storage area.
- Set attributes such as page size to better correspond with tables that will be stored in the storage area.

**storage-area-params-1**
**storage-area-params-2**
Parameters that control the characteristics of the storage area.

**ALLOCATION IS number-pages PAGES**
The number of database pages initially allocated to the storage area. SQL automatically extends the allocation to handle the storage requirements. Pages are allocated in groups of three (known as a *clump*). An ALLOCATION of 25 pages actually provides for 27 pages of data and subsequent expansion. The default is 400 pages.

**CACHE USING row-cache-name**
Assigns the named row cache to the specified storage area. All rows stored in this area, whether they consist of table data, segmented string data, or special rows such as index nodes, are cached if those rows fit in the cache.

If the row cache area does not exist, you must create the row cache area before terminating the CREATE DATABASE statement. For example:

```
SQL> CREATE DATABASE FILENAME test_db
cont> ROW CACHE IS ENABLED
cont> CREATE STORAGE AREA area1
cont>    CACHE USING test1
cont> CREATE CACHE test1
cont>    CACHE SIZE IS 100 ROWS
cont>    ROW LENGTH IS 200 BYTES;
```

Only one row cache is allowed for each storage area.

If you do not specify the CACHE USING clause or the NO ROW CACHE clause, NO ROW CACHE is the default for the storage area.

**NO ROW CACHE**
Specifies that a row cache area is not assigned to the specified storage area in the database. You cannot specify the NO ROW CACHE clause if you specify the CACHE USING clause.

Alter the storage area and name a row cache area with the CACHE USING clause to assign a row cache area to the storage area or to override the database default. Only one row cache area is allowed for each storage area.

If you do not specify the CACHE USING clause or the NO ROW CACHE clause, NO ROW CACHE is the default for the storage area.

**EXTENT ENABLED**
**EXTENT DISABLED**
Enables or disables extents. Extents are ENABLED by default.

You can encounter performance problems when creating hashed indexes in storage areas with the mixed page format if the storage area was created specifying the incorrect size for the area and if extents are enabled. By disabling extents, this problem can be diagnosed early and corrected to improve performance.

**EXTENT IS extent-pages PAGES**
**EXTENT IS (extension-options)**
Specifies the number of pages of each storage area file extent. See the description under the SNAPSHOT EXTENT argument.

**MINIMUM OF min-pages PAGES**
Specifies the minimum number of pages of each extent. The default is 100 pages.

**MAXIMUM OF max-pages PAGES**
Specifies the maximum number of pages of each extent. The default is 10,000 pages.

**PERCENT GROWTH IS growth**
Specifies the percent growth of each extent. The default is 20 percent growth.

**INTERVAL IS number-data-pages**
Specifies the number of data pages between SPAM pages in the storage area file, and thus the maximum number of data pages each SPAM page manages. The default, and also the minimum interval, is 216 data pages. The first page of each storage area is a SPAM page. The interval you specify determines where subsequent SPAM pages are to be inserted if there are enough data pages in the storage file to require more SPAM pages.

You cannot specify the INTERVAL storage area parameter unless you also explicitly specify PAGE FORMAT IS MIXED.

Oracle Rdb calculates the maximum INTERVAL size based on the number of blocks per page, and returns an error message if you exceed this value. For example, when the page size is 2 blocks, the maximum INTERVAL is 4008 pages. If you try to create a storage area with the INTERVAL set to 4009, Oracle Rdb returns the following error message:

```
%RDB-E-BAD_DPB_CONTENT, invalid database parameters in the database parameter
block (DPB)
-RDMS-F-SPIMAX, spam interval of 4009 is more than the Rdb maximum of 4008
-RDMS-F-AREA_NAME, area NEW
```

For more information about setting space area management parameters, see the *Oracle Rdb7 Guide to Database Maintenance.*

**LOCKING IS ROW LEVEL**
**LOCKING IS PAGE LEVEL**
Specifies if locking is at the page or row level for the storage area. This clause provides an alternative to requesting locks on records. Specifying a lock level when you create a storage area overrides the database default lock level. The default is ROW LEVEL, which is compatible with previous versions of Oracle Rdb.

When many records are accessed in the same area and on the same page, the LOCKING IS PAGE LEVEL clause reduces the number of lock operations performed to process a transaction; however, this is at the expense of reduced concurrency. Transactions that benefit most with page-level locking are of short duration and also access several database records on the same page.

Use the LOCKING IS ROW LEVEL if transactions are long in duration and lock many rows.

The LOCKING IS PAGE LEVEL clause causes fewer blocking ASTs and provides better response time and utilization of system resources. However, there is a higher contention for pages and increased potential for deadlocks.

Page-level locking is *never* applied to RDB$SYSTEM, either implicitly or explicitly, because the locking protocol can stall metadata users.

You cannot specify page-level locking on single-file databases.

**PAGE FORMAT IS UNIFORM**
**PAGE FORMAT IS MIXED**
Specifies the on-disk structure for the storage area.

- The default is PAGE FORMAT IS UNIFORM. A storage area with uniform page format is a file that is divided into groups of *n* pages, called **clumps**, where *n* equals the buffer size divided by the page size. Both buffer size and page size are user specified values. By default, the buffer size is 6 blocks, and the page size is 1024 bytes or 2 blocks long, resulting in clumps of three pages. The PAGE FORMAT IS UNIFORM argument creates a storage area file that is divided into clumps. A set of clumps forms a **logical area** that can contain rows from a single table only.

  Uniform page format storage areas generally give the best performance if the tables in the storage area are likely to be subject to a wide range of queries.

- The PAGE FORMAT IS MIXED argument creates a storage area with a format that allows rows from more than one table to reside on or near a particular page of the storage area file. This is useful for storing related rows from different tables on the same page of the data file. For storage areas subject to repeated queries that retrieve those related rows, a mixed page format can greatly reduce input/output overhead if the mix of rows on the page is carefully controlled. However, mixed page format storage areas degrade performance if the mix of rows on the page is not suited for the queries made against the storage area.

For more information on the relative advantages and disadvantages of uniform and mixed storage areas, see the *Oracle Rdb7 Guide to Database Maintenance*.

**PAGE SIZE IS page-blocks BLOCKS**
The size in blocks of each data page in the storage area. Page size is allocated in 512-byte blocks. The default is 2 blocks (1024 bytes). If your largest row is larger than approximately 950 bytes, allocate more blocks per page to prevent fragmented rows. If you specify a page size larger than the buffer size, an error message is returned.

**CHECKSUM CALCULATION IS ENABLED**
**CHECKSUM CALCULATION IS DISABLED**
This option allows you to enable or disable calculations of page checksums when pages are read from or written to the storage area files.

The default is ENABLED.

**Note**

Oracle Rdb recommends that you leave checksum calculations enabled; which is the default.

With current technology, it is possible that errors may occur that the checksum calculation can detect but that may not be detected by either the hardware, firmware, or software. Unexpected application results and database corruption may occur if corrupt pages exist in memory or on disk but are not detected.

Oracle Rdb recommends performing checksum calculations, except in the following specific circumstances:

- Your application is stable and has run without errors on the current hardware and software configuration for an extended period of time.
- You have reached maximum CPU utilization in your current configuration. Actual CPU utilization by the checksum calculation depends primarily on the size of the database pages in your database. The larger the database page, the more noticeable the CPU usage by the checksum calculation may become.

**Note**

Oracle Rdb recommends that you carefully evaluate the trade-off between reducing CPU usage by the checksum calculation and the potential for loss of database integrity if checksum calculations are disabled.

Oracle Rdb allows you to disable and, subsequently, re-enable checksum calculation without error. However, once checksum calculations have been disabled, corrupt pages may not be detected even if checksum calculations are subsequently re-enabled.

**SNAPSHOT CHECKSUM CALCULATION IS ENABLED**
**SNAPSHOT CHECKSUM CALCULATION IS DISABLED**
This option allows you to enable or disable calculations of page checksums when pages are read from or written to the snapshot files.

The default is ENABLED.

**Note**

Oracle Rdb recommends that you leave snapshot checksum calculations enabled, which is the default.

With current technology, it is possible that errors may occur that the snapshot checksum calculation can detect but that may not be detected by either the hardware, firmware, or software. Unexpected application results and database corruption may occur if corrupt pages exist in memory or on disk but are not detected.

Oracle Rdb recommends performing snapshot checksum calculations, except in the following specific circumstances:

- Your application is stable and has run without errors on the current hardware and software configuration for an extended period of time.
- You have reached maximum CPU utilization in your current configuration. Actual CPU utilization by the snapshot checksum calculation depends primarily on the size of the database pages in your database. The larger the database page, the more noticeable the CPU usage by the snapshot checksum calculation may become.

**Note**

Oracle Rdb recommends that you carefully evaluate the trade-off between reducing CPU usage by the snapshot checksum calculation and the potential for loss of database integrity if snapshot checksum calculations are disabled.

Oracle Rdb allows you to disable and, subsequently, re-enable snapshot checksum calculation without error. However, once snapshot checksum calculations have been disabled, corrupt pages may not be detected even if snapshot checksum calculations are subsequently re-enabled.

**SNAPSHOT ALLOCATION IS snp-pages PAGES**
Specifies the number of pages allocated for the snapshot file. The default is 100 pages.

**SNAPSHOT EXTENT IS extent-pages PAGES**
**SNAPSHOT EXTENT IS (extension-options)**
Specifies the number of pages of each snapshot or storage area file extent. The default extent for storage area files is 100 pages.

Specify a number of pages for simple control over the extension. For greater control, and particularly for multivolume databases, use the MIN, MAX, and PERCENT GROWTH extension options instead.

If you use the MIN, MAX, and PERCENT GROWTH parameters, you must enclose them in parentheses.

**SNAPSHOT FILENAME file-spec**
Provides a separate file specification for the snapshot file. The SNAPSHOT FILENAME argument can only be specified with multifile databases.

This argument lets you specify a different file name, device, or directory for the snapshot file created by the CREATE STORAGE AREA clause. Do not specify a file extension other than .snp to the file specification. Oracle Rdb assigns the extension .snp to the file specification, even if you specify an alternate extension.

If you omit the SNAPSHOT FILENAME argument, the snapshot file gets the same device, directory, and file name as the storage area file.

**THRESHOLDS ARE (val1 [,val2 [,val3] ])**
Specifies one, two, or three threshold values for mixed format pages. The threshold values represent a fullness percentage on a data page and establish three possible ranges of guaranteed free space on the data pages. When a data page reaches the percentage defined by a given threshold value, the space area management (SPAM) entry for the data page is updated to reflect the new fullness percentage and its remaining free space.

The default threshold values for mixed areas, if not specified, are (70,85,95), which indicates that the nominal record size should be used for SPAM threshold calculations. Oracle Rdb never stores a record on a page at the third threshold. The value you set for the highest threshold can be used to reserve space on the page for future record growth.

When only val1 is specified, this is equivalent to (val1, 100, 100). When val1 and val2 are specified, this is equivalent to (val1, val2, 100). The trailing, unspecified thresholds default to 100 percent. For example, THRESHOLDS ARE (40) would appear as (40, 100, 100).

You cannot specify the THRESHOLDS storage area parameter unless you also explicitly specify PAGE FORMAT IS MIXED.

For more information about setting space area management parameters, see the *Oracle Rdb7 Guide to Database Maintenance*.

**WRITE ONCE**
The WRITE ONCE option of the storage-area-params clause permits you to create a storage area that contains only a segmented string in a format that can be stored on a write-once, read-many (WORM) optical device.

Oracle Rdb permits the storing of many write-once list segments on one write-once page, resulting in better write-once space usage. This improves storage performance because the storage algorithm reduces I/O due to more compact storage.

The following restrictions apply to the WRITE ONCE option:

- You cannot write data other than segmented strings to a write-once storage area. SQL issues an error message if you try to create a storage map that stores data other than segmented strings in a write-once storage area.
- When you create a storage area on WORM media, you must specify that the snapshot area remains on a read/write device; do not give a snapshot file the WRITE ONCE attribute.
- If you specify the WRITE ONCE option when storing a segmented string, database keys are not compressed. For more information on database key compression, see the *Oracle Rdb7 Guide to Database Maintenance*.
- WORM areas do not use SPAM pages. However, to assist moving data back to non-WORM devices on which SPAM pages must be built again, space is still allocated for them. Because SPAM pages are essential in uniform areas, write-once storage areas cannot be of uniform format and, therefore, are required to be of mixed format.
- You can use the PAGE SIZE IS clause of the CREATE DATABASE statement to set the default page size for a storage area. To optimize storage, always specify an even number of blocks per page for a write-once storage area.
- Oracle Rdb does not support magnetic media for storing write-once storage areas.
- After you move a storage area to or from WORM media, back up your database completely and start a new after-image journal file. For more information on backup and recovery procedures with write-once storage areas, see the *Oracle Rdb7 Guide to Database Maintenance*.
- The storage algorithm does not attempt to compute the best fit for write-once list segments.
- The storage algorithm does not allow write-once storage by different users to be on the same write-once page.
- If the number of buffers is small, a write-once page that is only partially full may be flushed out of the buffer pool (and hence written to disk) as part of the usual buffer replacement policy.

**JOURNAL IS ENABLED**
**JOURNAL IS DISABLED**
Specifies whether or not WRITE ONCE areas are written to the AIJ file.

Disabling the journaling attribute on WRITE ONCE areas is beneficial because after-image journaling on storage media can slow the loading of large images or exceed storage area availability.

However, if there is a failure of the storage media, there may be loss of space or, more important, loss of information. In the case of a magnetic disk failure, the database is restored from an earlier backup and the AIJ records are applied to the restored database. There is no loss of information in this case, but there could be loss of space because list of byte varying data written before the failure is not referenced by the existing data rows, and these list column values take up space on the write-once media that cannot be reused.

In the case of a WORM device failure, there can be loss of information because the existing data rows reference list column data that is no longer available. For example, if 120 pages were allocated in the WRITE ONCE area and 100 pages had data written to the them, and the last backup was done when the area had 50 pages of information, any data on pages 51 to 120 is lost if there is a failure of the WORM device. Pages 51 to 120 are inaccessible. The RMU Repair command can be used to repair rows that reference missing list column data. For more information, see the *Oracle Rdb7 Guide to Database Maintenance* and the *Oracle RMU Reference Manual*.

Remember, the write-once storage area must be of mixed format.

The default is JOURNAL IS ENABLED.

## Usage Notes

- You cannot use the CREATE STORAGE AREA clause with single-file databases. The presence or absence of a CREATE STORAGE AREA clause in a CREATE DATABASE statement is what determines whether the database is single file or multifile. SQL creates a multifile database only when the CREATE DATABASE statement includes at least one CREATE STORAGE AREA clause.
- The CREATE STORAGE AREA clause does not control which tables will actually be associated with the storage area. The CREATE STORAGE MAP statement controls which parts of which tables are stored in a particular storage area file. For information about storing lists, see the CREATE STORAGE MAP Statement.

- If the LOCKING IS PAGE LEVEL or LOCKING IS ROW LEVEL clause is specified at the database level (using the ALTER DATABASE or CREATE DATABASE statements), all storage areas are affected (with the exception of RDB$SYSTEM which is always set to row-level locking). If specified at the storage area level (using the CREATE STORAGE AREA clause), only the specified storage area attributes are affected.
- Page locks are held until the end of the transaction.
- The buffer pool should be sized so that it is large enough to contain all the pages that are required for a transaction. Use the RMU Show Statistics command to determine whether or not the buffer pool is sized correctly during a transaction. See the *Oracle RMU Reference Manual* and the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information.
- If you specify the WRITE ONCE (JOURNAL IS DISABLED) clause, a database that is recovered to a time prior to all transactions being committed causes old list of byte varying data to be visible again. If the database is recovered using a shadow copy, access to some list of byte varying columns return an exception to indicate that old data is present on the write-once media.
- The CACHE USING clause assigns a row cache area to a storage area.

## Examples

Example 1: Defining a multifile database

This example shows the definition of a database and storage areas for a multifile database on OpenVMS. (On Digital UNIX, only the file specifications would be different from the following example.)

See the CREATE STORAGE MAP Statement for an example of the CREATE STORAGE MAP statement that associates particular tables with the storage areas created in this example.

```
SQL> -- Note that there is no semicolon before
SQL> -- the first CREATE STORAGE AREA clause.
SQL> CREATE DATABASE ALIAS MULTIFILE_EXAMPLE
cont>   FILENAME 'DB_DATA01:[DB.DATA]MULTIFILE_EXAMPLE'
cont> CREATE STORAGE AREA EMPID_LOW
cont>   FILENAME 'DB_DATA02:[DB.DATA]EMPID_LOW'
cont>   ALLOCATION IS 10 PAGES
cont>   -- Notice that the snapshot file resides on a
cont>   -- different disk than the storage area file. This
cont>   -- strategy reduces disk input/output bottlenecks:
cont>   SNAPSHOT FILENAME 'DB_SNAP03:[DB.SNAP]EMPID_LOW'
cont>   SNAPSHOT ALLOCATION IS 10 PAGES
cont> --
cont> CREATE STORAGE AREA EMPID_MID
cont>   FILENAME 'DB_DATA04:EMPID_MID'
cont>   ALLOCATION IS 10 PAGES
cont>   SNAPSHOT FILENAME 'DB_SNAP05:[DB.SNAP]EMPID_MID'
cont>   SNAPSHOT ALLOCATION IS 10 PAGES
cont> --
cont> CREATE STORAGE AREA EMPID_OVER
cont>   FILENAME 'DB_DATA06:[DB.DATA]EMPID_OVER'
cont>   ALLOCATION IS 10 PAGES
cont>   SNAPSHOT FILENAME 'DB_SNAP07:[DB.SNAP]EMPID_OVER'
cont>   SNAPSHOT ALLOCATION IS 10 PAGES
cont> --
cont> CREATE STORAGE AREA HISTORIES
cont>   FILENAME 'DB_DATA02:[DB.DATA]HISTORIES'
cont>   ALLOCATION IS 10 PAGES
cont>   SNAPSHOT FILENAME 'DB_SNAP03:[DB.SNAP]HISTORIES'
cont>   SNAPSHOT ALLOCATION IS 10 PAGES

cont> --
cont> CREATE STORAGE AREA CODES
cont>   FILENAME 'DB_DATA04:[DB.DATA]CODES'
cont>   ALLOCATION IS 10 PAGES
cont>   SNAPSHOT FILENAME 'DB_SNAP05:[DB.SNAP]CODES'
cont>   SNAPSHOT ALLOCATION IS 10 PAGES
cont> --
cont> CREATE STORAGE AREA EMP_INFO
cont>   FILENAME 'DB_DATA08:[DB.DATA]EMP_INFO'
cont>   ALLOCATION IS 10 PAGES
cont>   SNAPSHOT FILENAME 'DB_SNAP09:[DB.SNAP]EMP_INFO'
cont>   SNAPSHOT ALLOCATION IS 10 PAGES
cont> --
cont> -- End the CREATE DATABASE statement:
cont> ;
```

Example 2:

This example shows how to set page-level and row-level locking on storage areas from both the database level and from the storage area level.

```
SQL> CREATE DATABASE FILENAME sample
cont>    LOCKING IS PAGE LEVEL
cont> --
cont> -- All storage areas will default to page-level locking unless
cont> -- explicitly set to row-level locking.
cont> --
cont> CREATE STORAGE AREA RDB$SYSTEM
cont>    FILENAME sample_system
cont> --
cont> -- You cannot specify page-level locking on RDB$SYSTEM.  RDB$SYSTEM
cont> -- always defaults to row-level locking.
cont> --
cont> CREATE STORAGE AREA HASH_AREA
cont>    FILENAME sample_hash
cont>    PAGE FORMAT IS MIXED
cont> --
cont> -- HASH_AREA defaultS to page-level locking.
cont> --
cont> CREATE STORAGE AREA DATA_AREA
cont>    FILENAME sample_data
cont>    LOCKING IS ROW LEVEL
cont> --
cont> -- DATA_AREA is explicitly set to row-level locking.
cont> --
cont> ;
SQL> SHOW STORAGE AREAS (ATTRIBUTES) *
Storage Areas in database with filename sample

     RDB$SYSTEM
         List storage area.
         Access is:      Read write
         Page Format:    Uniform
         Page Size:      2 blocks
         .
         .
         .
         Extent :       Enabled
         Locking is Row Level
     HASH_AREA
         Access is:      Read write
         Page Format:    Mixed
         Page Size:      2 blocks
         .
         .
         .
         Extent :       Enabled
         Locking is Page Level

     DATA_AREA
         Access is:      Read write
         Page Format:    Uniform
         Page Size:      2 blocks
         .
         .
         .
         Extent :       Enabled
         Locking is Row Level
```

See the SHOW Statement for information on the SHOW STORAGE AREAS statement.

Example 3: Disabling the journaling mode on write-once storage areas

```
SQL> CREATE DATABASE FILENAME test
cont> CREATE STORAGE AREA RDB$SYSTEM
cont>       FILENAME test_system
cont> CREATE STORAGE AREA LIST_AREA
cont>       FILENAME list_system
cont>       PAGE FORMAT IS MIXED
cont>       WRITE ONCE (JOURNAL IS DISABLED);
SQL> SHOW STORAGE AREA (ATTRIBUTES) LIST_AREA;

     LIST_AREA
         Page Format:    Mixed
         Page Size:      2 blocks
         .
         .
         .
         Extent :       Enabled
         Locking is Row Level
         Write Once (Journal is Disabled)
```

Example 4: Creating and assigning a row cache area to a storage area

```
SQL> CREATE DATABASE FILENAME sample_db
cont>   RESERVE 5 CACHE SLOTS
cont>   ROW CACHE IS ENABLED
cont>   DEFAULT STORAGE AREA IS area1
cont> CREATE CACHE cache1
cont>   CACHE SIZE IS 1000 ROWS
cont>   ROW LENGTH IS 1000 BYTES
cont> CREATE STORAGE AREA area1
cont>   CACHE USING cache1;
SQL> SHOW CACHE cache1

     CACHE1
        Cache Size:            1000 rows
        Row Length:            1000 bytes
        Row Replacement:       Enabled
        Shared Memory:         Process
        Large Memory:          Disabled
        Window Count:          100
        Reserved Rows:         20
        Sweep Rows:            3000
        No Sweep Thresholds
        Allocation:            100 blocks
        Extent:                100 blocks
SQL> SHOW STORAGE AREA area1;

     AREA1
         Access is:      Read write
         Page Format:    Uniform
         Page Size:      2 blocks
         Area File:      SQL_USER1:[DAY.V70]AREA1.RDA;3
         Area Allocation:          402 pages
         Area Extent Minimum:      99 pages
         Area Extent Maximum:      9999 pages
         Area Extent Percent:      20 percent
         Snapshot File:  SQL_USER1:[DAY.V70]AREA1.SNP;3
         Snapshot Allocation:      100 pages
         Snapshot Extent Minimum:  99 pages
         Snapshot Extent Maximum:  9999 pages
         Snapshot Extent Percent:  20 percent
         Extent :       Enabled
         Locking is Row Level
         Using Cache CACHE1

Database objects using Storage Area AREA1:
Usage            Object Name                      Map / Partition
---------------- -------------------------------- -----------------
Default Area
```

## CREATE STORAGE MAP Statement

Associates a table with one or more storage areas in a multifile database. The CREATE STORAGE MAP statement specifies a **storage map** that controls which lists or rows of a table are stored in which storage areas.

In addition to creating storage maps, the CREATE STORAGE MAP statement has options that control:

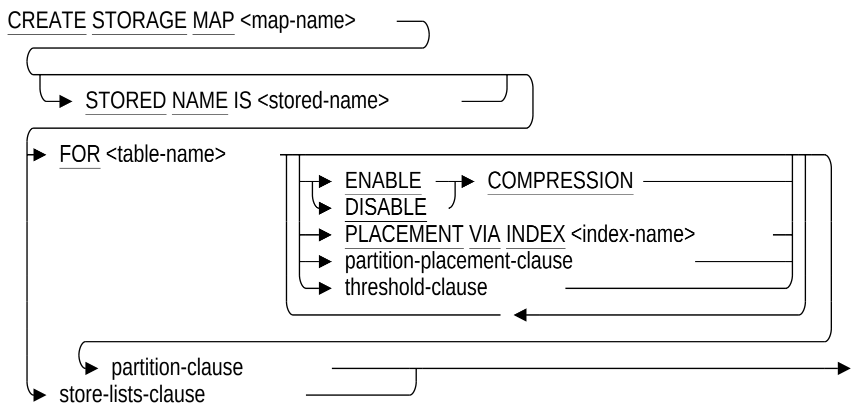
- Which index the database system uses when inserting rows in the table
- Whether or not the rows of the table are stored in a compressed format
- Whether or not partitioning keys can be modified.
- Whether the table is partitioned vertically, horizontally, or both.

### Environment

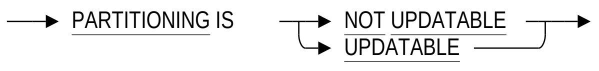
You can use the CREATE STORAGE MAP statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

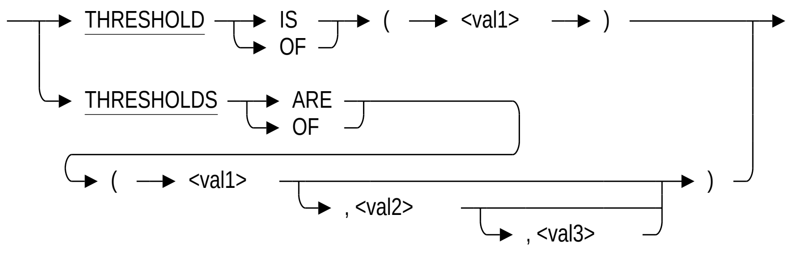
### Format


CREATE STORAGE MAP <map-name>
STORED NAME IS <stored-name>
FOR <table-name>
ENABLE
DISABLE
COMPRESSION
PLACEMENT VIA INDEX <index-name>
partition-placement-clause
threshold-clause
partition-clause
store-lists-clause

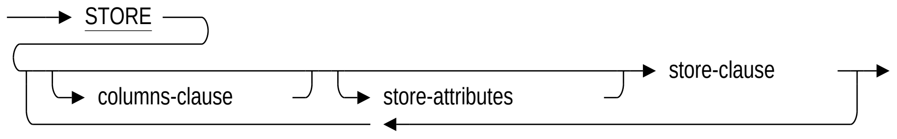
partition-placement-clause =



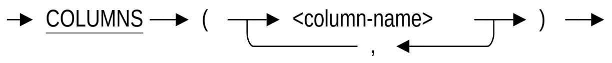
threshold-clause =



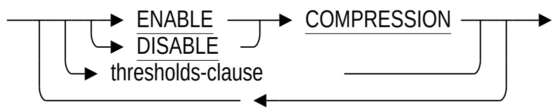
partition-clause =



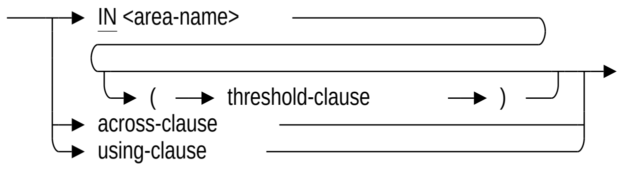
columns-clause =



store-attributes =

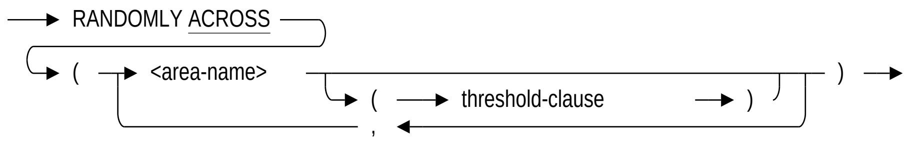


store-clause =

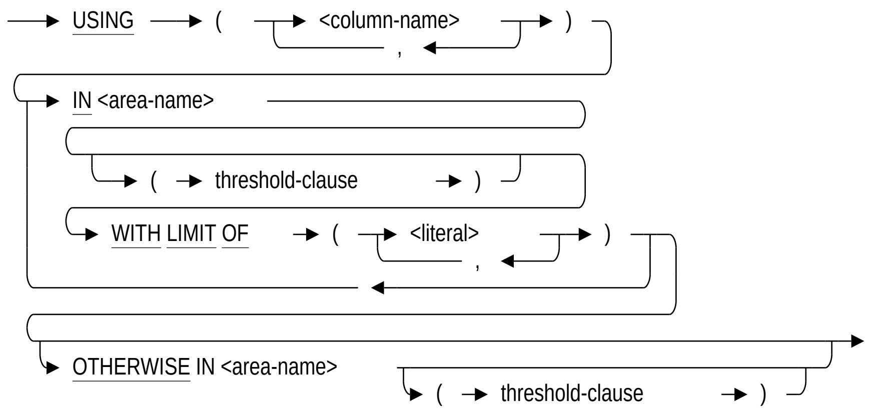


## CREATE STORAGE MAP Statement

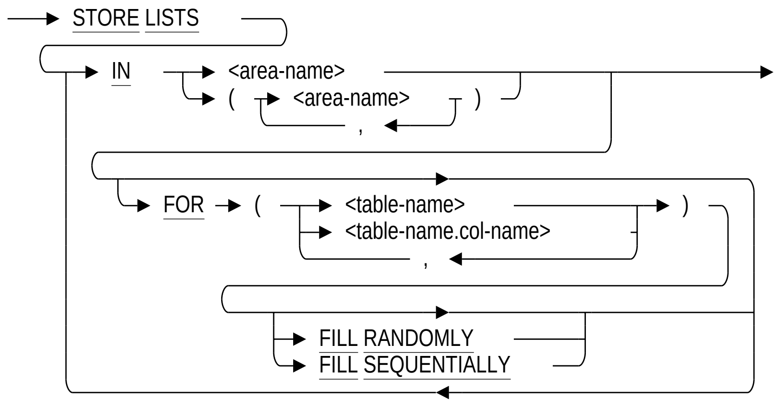
across clause =



using-clause =



store-lists-clause =



## Arguments

**STORAGE MAP map-name**
Specifies the name of the storage map you want to create. The name cannot be the same as any other definition in the database.

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access a storage map created in a multischema database. The stored name allows you to access multischema definitions using interfaces, such as Oracle RMU, the Oracle Rdb management utility, that do not recognize multiple schemas in one database. You cannot specify a stored name for a storage map in a database that does not allow multiple schemas. For more information on stored names, see Section 2.2.4.

**FOR table-name**
Specifies the table to which this storage map applies. The named table must already be defined and cannot have a storage map associated with it.

**ENABLE COMPRESSION**
**DISABLE COMPRESSION**
Specifies whether the rows for the table are compressed or uncompressed when stored. The default is ENABLE COMPRESSION.

You can enable compression to conserve disk space, but there is additional CPU overhead for inserting and retrieving compressed rows.

**PLACEMENT VIA INDEX index-name**
Directs the database system to store a column in a way that optimizes access to that column by the indicated path. Oracle Rdb chooses a target page for any columns being stored by rules that take into account the type of index named (sorted or hashed), the type of storage areas involved (uniform or mixed), and how indexes and tables are assigned to storage areas.

For a hashed index, Oracle Rdb calculates the page containing the hashed index node that points to the column. If that page is within the same storage area in which the column will be stored, it is used as the target page for storing the column. If that page is not within the same storage area in which the column is to be stored, Oracle Rdb chooses a target page in the same relative position within the appropriate storage area (if it is a mixed storage area) or a page in a clump reserved for that table (if it is a uniform storage area).

For a sorted index, Oracle Rdb finds the database key of the next lowest row to the one being stored and uses the page number in the database key as the target page.

**PARTITIONING IS NOT UPDATABLE**
Specifies that the value of the partitioning key cannot be modified and that the row is always stored in the storage area based on the partitioning criteria in the STORE USING clause. The partitioning key is the column or list of columns specified in the STORE USING clause.

Specifying the PARTITIONING IS NOT UPDATABLE clause allows Oracle Rdb to quickly retrieve data because the partitioning criteria can be used when optimizing the query.

To update columns that are partitioning keys in a NOT UPDATABLE storage map, you must delete the rows and then reinsert the rows to ensure that they are placed in the correct location.

If you specify the PARTITIONING clause, you must also specify the STORE USING clause when defining a storage map.

If the PARTITIONING clause is not specified, UPDATABLE is the default.

See the *Oracle Rdb7 Guide to Database Design and Definition* for more information regarding partitioning.

**PARTITIONING IS UPDATABLE**
Specifies that the partitioning key can be modified. The partitioning key is the column or list of columns specified in the STORE USING clause.

If you modify a row in an UPDATABLE storage map, the row is not moved to a different storage area even if the new value of the partitioning key is not within the limits of original storage area. As a result, Oracle Rdb must consider all storage areas specified in the STORE USING clause when retrieving a row.

If you specify the PARTITIONING clause, you must also specify the STORE USING clause when defining a storage map.

If the PARTITIONING clause is not specified, UPDATABLE is the default.

See the *Oracle Rdb7 Guide to Database Design and Definition* for more information regarding partitioning.

**threshold-clause**
Specifies one, two, or three default threshold values for logical areas in storage areas with uniform format pages. The threshold values (val1, val2, and val3) represent a fullness percentage on a data page and establish three possible ranges of guaranteed free space on the data pages. When a data page reaches the percentage defined by a given threshold value, the space area management (SPAM) entry for the data page is updated to reflect the new fullness percentage and its remaining free space.

Oracle Rdb never stores a record at the third threshold. The value you set for the highest threshold can be used to reserve space on the page for future record growth.

When only val1 is specified, this is equivalent to (val1, 100, 100). When val1 and val2 are specified, this is equivalent to (val1, val2, 100). The trailing, unspecified thresholds default to 100 percent. For example, THRESHOLDS ARE (40) would appear as (40, 100, 100).

If no thresholds are specified for the area, the default is (0,0,0). This causes the SPAM algorithm to set thresholds based on the nominal record length for the logical area; for example, the node size for the index or the uncompressed length of the row for a table.

If you use data compression, you should use logical area thresholds.

Although a threshold clause associated with a particular area is enclosed in parentheses, the default threshold clause for the storage map is not.

When the threshold clause is part of an IN area-name clause, you can use the form THRESHOLD or THRESHOLDS.

You cannot specify the thresholds for the storage map attribute for any area that is a mixed page format. If you have a mixed page format, set the thresholds for the storage area using the ADD STORAGE AREA or CREATE STORAGE AREA clause of the ALTER DATABASE, CREATE DATABASE, or IMPORT statements.

**partition-clause**
Defines vertical partitioning, horizontal partitioning, or both for the specified table.

**Horizontal partitioning** means that you divide the rows of the table among storage areas according to data values in one or more columns. **Vertical partitioning** means that you divide the columns of the table among storage areas. A given storage area will then contain only some of the columns of a table.

Vertical partitioning reduces disk I/O operations by placing frequently used data in one area, so that you can read and update those portions of the table in a single disk I/O operation.

See the *Oracle Rdb7 Guide to Database Design and Definition* for more information regarding partitioning.

**STORE COLUMNS (column-name)**
Specifies the columns that are to be vertically partitioned. The columns listed are also called **partitioning keys**.

**ENABLE COMPRESSION**
**DISABLE COMPRESSION**
Specifies whether the rows for the partition are compressed or uncompressed when stored. You can enable or disable compression on each vertical partition. If you omit this clause, the default compression is that which was specified for the storage map before the STORE COLUMNS clause.

You enable compression to conserve disk space, but there is additional CPU overhead for inserting and retrieving compressed rows.

**thresholds-clause**
Specifies one, two, or three threshold values for a logical area in a storage area with uniform format pages. For more information about threshold values, see the first threshold-clause description in this Arguments list.

**store-clause**
The storage map definition. The store-clause in a CREATE STORAGE MAP statement lets you specify which storage area files are used to store rows from the table.

- All rows of a table can be associated with a single storage area.
- Rows of a table can be randomly distributed among several storage areas.
- Rows of a table can be systematically distributed, or partitioned, among several storage areas by specifying upper limits on the values for a column in a particular storage area. This is called horizontal partitioning.
- Columns of a table can be partitioned among storage areas. This is called vertical partitioning.

If you omit the storage map definition, the default is to store all the rows for a table in the main RDB$SYSTEM storage area.

**STORE IN area-name**
Associates the table directly with a single storage area. All rows in the table are stored in the area you specify.

**across-clause**
Associates the table with two or more storage areas.

**STORE RANDOMLY ACROSS (area-name)**
As rows are inserted in the table, they are distributed randomly across the storage areas named in the list. You must name at least two storage areas in this clause.

**threshold-clause**
Specifies one, two, or three threshold values for a logical area in a storage area with uniform format pages. For more information about threshold values, see the first threshold-clause description in this Arguments list.

**using-clause**
Specifies columns whose values are used as limits for partitioning the table horizontally across multiple storage areas.

**STORE USING (column-name) IN area-name**
The database system compares values in the columns to the values in the WITH LIMIT OF clause to determine placement of rows inserted into the table. For instance, a storage map with the clause STORE USING (X,Y,Z) IN AREA1 WITH LIMIT OF (1,2,3) means that a row must meet these criteria to be stored in AREA1:

$$(X < 1)\ OR\ ((X = 1)\ AND\ ((Y < 2)\ OR\ ((Y = 2)\ AND\ (Z \leq 3))))$$

**WITH LIMIT OF (literal)**
Specifies the maximum values that the columns named in the USING clause can have when rows are initially stored in the specified storage area. Repeat this clause to partition the rows of a table among multiple storage areas.

The number of literals listed must be the same as the number of columns in the USING clause. The data type of the literals must agree with the data type of the column. For character columns, enclose the literals in single quotation marks.

The values in the WITH LIMIT OF clause only affect placement of rows when they are initially stored. If UPDATE statements change data in a row so that values in columns named in the USING clause exceed values specified in the WITH LIMIT OF clause, the row is not moved into a different storage area.

When you create a multisegmented index using multisegmented keys and use the STORE USING . . . WITH LIMIT OF clauses, if the values for the first key are all the same, then set the limit for the first key at that value. This ensures that the value of the second key determines the storage area in which each row will be stored.

**OTHERWISE IN area-name**
For partitioned storage maps only, specifies the storage area that is used as the overflow partition. An **overflow partition** is a storage area that holds any values that are higher than those specified in the WITH LIMIT TO clause. An overflow partition holds those values that “overflow” the partitions that have specified limits.

**STORE LISTS IN area-name**

Directs the database system to store the lists from tables in a specified storage area or in a set of areas. You can create only one storage map for lists within each database.

You must specify a default storage area for lists in the STORE LISTS clause. The default list storage area contains lists from system tables as well as lists not directed elsewhere by the STORE LISTS clause. You can also use the LIST STORAGE AREA clause of the CREATE DATABASE statement to specify a default storage area for lists. If you do not use the STORE LISTS clause and do not specify a list storage area in the CREATE DATABASE statement, Oracle Rdb uses RDB$SYSTEM as the default list storage area. The following example directs Oracle Rdb to place all lists in the LISTS storage area unless otherwise specified in a storage map:

```
SQL> CREATE DATABASE FILENAME mf_personnel
SQL> LIST STORAGE AREA IS LISTS
SQL> CREATE STORAGE AREA LISTS;
```

The accompanying storage map statement must also specify the LISTS storage area as the default storage area.

```
SQL> CREATE STORAGE MAP LISTS_MAP
cont> STORE LISTS IN LISTS1 FOR (EMPLOYEES.RESUME)
cont>             IN LISTS;
```

You can use an area set to specify that data is to be distributed across several areas. The following example shows how you can store data in three storage areas (LISTS1, LISTS2, and LISTS3) for two different columns in TABLE1. The default list storage area is LISTS1.

```
CREATE STORAGE MAP LISTS_MAP
       STORE LISTS IN (LISTS1,LISTS2,LISTS3) FOR (TABLE1.COL1,TABLE1.COL2)
       IN LISTS1;
```

You can store lists from different tables in the same area. The following example shows how you can store data from TABLE1, TABLE2, and TABLE3 in the LISTS storage area. The default list storage area is RDB$SYSTEM.

```
SQL> CREATE STORAGE MAP LISTS_MAP  -- to direct the list data to area LISTS
cont>  STORE LISTS IN LISTS FOR (TABLE1, TABLE2, TABLE3)
cont>     IN RDB$SYSTEM;
```

Alternatively, you can store lists from each table in unique areas. The following example shows list data from TABLE1 being stored in the LISTS1 storage area and list data from TABLE2 being stored in the LISTS2 storage area. The default list storage area is RDB$SYSTEM.

```
CREATE STORAGE MAP LISTS_MAP
       STORE LISTS IN LIST1 FOR (TABLE1)
                   IN LIST2 FOR (TABLE2)
                   IN RDB$SYSTEM;
```

You can also specify that different columns from the same table go into different areas. The following example shows data from different columns in TABLE1 being stored in either LISTS1 or LISTS2. The default list storage area is RDB$SYSTEM.

```
CREATE STORAGE MAP LISTS_MAP
       STORE LISTS IN LISTS1 FOR (TABLE1.COL1)
                   IN LISTS2 FOR (TABLE1.COL2)
                   IN RDB$SYSTEM;
```

**FOR (table-name)**
Specifies the table or tables to which this storage map applies. The named table must already be defined. If you want to store lists of more than one table in the storage area, separate the names of the tables with commas. For each area, you can specify one FOR clause and list of table names.

**FOR (table-name.col-name)**
Specifies the name of the table and column containing the list to which this storage map applies. Separate the table name and the column name with a period (.). The named table and column must already be defined. If you want to store multiple lists in the storage area, separate the table name and column name combinations with commas. For each area, you can specify one FOR clause and a list of column names.

**FILL RANDOMLY**
**FILL SEQUENTIALLY**
Specifies whether to fill the area set randomly or sequentially. Specifying FILL RANDOMLY or FILL SEQUENTIALLY requires a FOR clause. When a storage area is filled, it is removed from the list of available areas. Oracle Rdb does not attempt to store any more lists in that area during the current database attach. Instead, Oracle Rdb starts filling the next specified area.

When a set of areas is filled sequentially, Oracle Rdb stores lists in the first specified area until that area is filled. Use sequential filling when storing lists in write-once storage areas in a jukebox environment to avoid excess swapping of platters. In a jukebox environment, the filled storage area is marked with a FULL flag and the platter on which the area resides is no longer swapped in.

If the set of areas is filled randomly, lists are stored across multiple areas. This is the default. Random filling is intended for read/write media, which will benefit from the I/O distribution across the storage areas.

The keywords FILL RANDOMLY and FILL SEQUENTIALLY can only be applied to areas contained within an area list.

## Usage Notes

- You must specify either a STORE clause, a PLACEMENT clause, or a COMPRESSION clause in a CREATE STORAGE MAP statement.
- If you specify multiple storage areas in a CREATE STORAGE MAP statement, they must have the same format; you cannot specify both MIXED and UNIFORM format storage areas in the same storage map.
- You cannot create more than one map for the rows from a given table, but you can create one map for that table's rows and a separate map for that table's lists.
- If you repeat a column or table in the storage map with a different area, then all columns of data type LIST OF BYTE VARYING are stored randomly across the specified areas, unless you specify SEQUENTIAL storage.
- You can mix horizontal and vertical partitioning in a single storage map.
- You cannot delete a list storage map from the database.
- For write-once storage areas, lists are stored in the first area until that area is full. A FULL flag is then set on the area; this FULL flag is displayed by the RMU Dump Header command. Read/write areas do not get marked FULL.

  The RMU Backup, RMU Move_Area, RMU Alter, and RMU Copy_Database commands clear the FULL flag because these operations move the data to a new device that may have more available space.
- At the current time, you can only specify one PLACEMENT VIA INDEX clause per storage map.
- The CREATE STORAGE MAP statement fails when both of the following circumstances are true:
  - The database to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The database was declared using the FILENAME argument.

Under these circumstances, the statement fails with the following error when you issue it:

```
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-CDDISREQ, CDD required for metadata updates is not being maintained
```

- Attempts to create a storage map fail if that storage map or its affected table is involved in a query at the same time. Users must detach from the database with a DISCONNECT statement before you can create the storage map. When Oracle Rdb first accesses an object such as the table, a lock is placed on that object and not released until the user exits the database. If you attempt to update this object, you get a LOCK CONFLICT ON CLIENT message due to the other user's access to the object.

  Similarly, while you create a storage map, users cannot execute queries involving that storage map until you have completed the transaction with a COMMIT or ROLLBACK statement for the CREATE statement. The user receives a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system tables. (System tables contain information about objects in the database.) Therefore, attempts to update an object locks out attempts to query that object. These locks are held until the COMMIT or ROLLBACK statement of the DDL operation completes.

  The WAIT/NOWAIT clause of the SET TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify SET TRANSACTION WAIT for the metadata update transaction, you will get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked
resource
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
SQL>
```

  However, a user's query will wait for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT in the SET TRANSACTION statement.

- You cannot execute the CREATE STORAGE MAP statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on the RDB$SYSTEM storage area.

- If you include a comment after the STORE clause in the CREATE STORAGE MAP statement, the comment is included in the storage map and table information.

  ```
  SQL> CREATE DATABASE FILENAME t
  cont> CREATE STORAGE AREA a
  cont> CREATE STORAGE AREA b
  cont> CREATE TABLE T1 (A CHAR(15), B INT)
  cont> CREATE STORAGE MAP M FOR T1 STORE USING (B) IN a
  cont> WITH LIMIT OF (5)
  cont> OTHERWISE IN RDB$SYSTEM
  cont> -- TEST CREATE STORAGE MAP
  cont> ;
  SQL> SHOW STORAGE MAP M
       M
   For Table:              T1
   Compression is:         ENABLED
   Store clause:           STORE USING (B) IN A
       WITH LIMIT OF (5)
       OTHERWISE IN RDB$SYSTEM
       -- TEST CREATE STORAGE MAP
  ```

  Use caution when using the STORE clause.

- If a storage map does not contain an overflow partition (defined by the OTHERWISE clause), you can add new partitions to the storage map without reorganizing the storage areas. For example:

  ```
  SQL> ALTER STORAGE MAP EMP_MAP
  cont>   STORE USING (EMPLOYEE_ID)
  cont>      IN PERSONNEL_1 WITH LIMIT OF ('00399')
  cont>      IN PERSONNEL_2 WITH LIMIT OF ('00699')
  cont>      IN PERSONNEL_3 WITH LIMIT OF ('10000')
  cont>      IN PERSONNEL_4 WITH LIMIT OF ('10399');
  SQL>
  ```

  Because the original storage map did not contain an OTHERWISE clause, you do not need to reorganize the storage areas.

  For more information, see the *Oracle Rdb7 Guide to Database Design and Definition* and the *Oracle Rdb7 Guide to Database Performance and Tuning*.

- If a storage map contains an overflow partition and you want to alter the storage map to change the overflow partition to a partition defined with the WITH LIMIT OF clause, you must use the REORGANIZE clause if you want existing data that is stored in the overflow partition moved to appropriate storage area. For example:

```
SQL> ALTER STORAGE MAP JH_MAP
cont>   STORE USING (EMPLOYEE_ID)
cont>      IN PERSONNEL_1 WITH LIMIT OF ('00399')
cont>      IN PERSONNEL_2 WITH LIMIT OF ('00699')
cont>      IN PERSONNEL_3 WITH LIMIT OF ('10000')
cont>      IN PERSONNEL_4 WITH LIMIT OF ('10399')
cont>   REORGANIZE;
SQL>
```

- If you attempt to insert values that are out of range of the storage map, you receive an error similar to the following:

  ```
  %RDMS-E-EXCMAPLIMIT, exceeded limit on last partition in storage map for
  EMPLOYEES
  ```

  Your applications should include code that handles this type of error.
- Oracle Rdb checks to ensure that list maps are not created on system tables. This check can only be done on data definition statements executed after an ATTACH statement. This check cannot be done when an attach is performed by the CREATE DATABASE or IMPORT statements because the map is created before the referenced list objects exist.
- You can create a storage map for an existing table that contains data. However, the following restrictions apply:
  - The storage map must be a simple map that references only the default storage area and represents the default mapping for the table.
  - You cannot change the thresholds or compression for the table.
  - You cannot specify the PLACEMENT VIA INDEX clause.
  - The storage map cannot be vertically partitioned.

  Once the storage map is created, you can use the ALTER STORAGE MAP statement to reorganize the table as needed.
- You cannot alter a vertically partitioned storage map once it is defined.
- Columns not specified in the using-clause are mapped to the final storage area.
- You must specify the columns-clause to vertically partition a storage map.
- If you are not vertically partitioning a storage map, only one store-clause is allowed in the storage map definition.

## Examples

Example 1: Defining storage maps for a multifile database

This example shows the definition of storage maps for a multifile database. The tables named in the CREATE STORAGE MAP statements have the same definitions as those in the sample database. See the CREATE STORAGE AREA Clause for an example of a CREATE DATABASE statement with CREATE STORAGE AREA clauses that create the storage areas referred to in this example.

```
SQL> -- Declare the database as the default:
SQL> ATTACH 'FILENAME multifile_example';
SQL> --
SQL> CREATE STORAGE MAP EMPLOYEE_MAP FOR EMPLOYEES
cont> STORE USING (EMPLOYEE_ID)
cont>   IN EMPID_LOW WITH LIMIT OF ('00200')
cont>   IN EMPID_MID WITH LIMIT OF ('00500')
cont>   OTHERWISE IN EMPID_OVER;
SQL> --
SQL> CREATE STORAGE MAP RESUME_MAP
cont> STORE LISTS IN EMP_INFO FOR (TABLE1, TABLE2, TABLE3)
cont>       IN RDB$SYSTEM;
SQL> --
SQL> CREATE STORAGE MAP JOB_HISTORY_MAP FOR JOB_HISTORY
cont> STORE IN HISTORIES;
SQL> --
SQL> CREATE STORAGE MAP SALARY_HISTORY_MAP FOR SALARY_HISTORY
cont> STORE IN HISTORIES;
SQL> --
SQL> CREATE STORAGE MAP JOBS_MAP FOR JOBS
cont> STORE IN CODES;
SQL> --
SQL> CREATE STORAGE MAP DEPARTMENTS_MAP FOR DEPARTMENTS
cont> STORE IN CODES;
SQL> --
SQL> CREATE STORAGE MAP COLLEGES_MAP FOR COLLEGES
cont> STORE IN CODES;
SQL> --
SQL> CREATE STORAGE MAP DEGREES_MAP FOR DEGREES
cont> STORE IN EMP_INFO;
SQL> --
SQL> CREATE STORAGE MAP WORK_STATUS_MAP FOR WORK_STATUS
cont> STORE IN HISTORIES;
SQL> --
SQL> --
SQL> -COMMIT;
SQL> --
```

Example 2: Defining storage maps that place and override thresholds on uniform storage areas

```
SQL> CREATE DATABASE FILENAME birdlist
cont>     CREATE STORAGE AREA AREA1
cont>     CREATE STORAGE AREA AREA2
cont>     CREATE STORAGE AREA AREA3
cont>     CREATE STORAGE AREA AREA4
cont>     CREATE TABLE SPECIES
cont>   ( GENUS         CHAR (30),
cont>     SPECIES       CHAR (30),
cont>     COMMON_NAME   CHAR (40),
cont>     FAMILY_NUMBER  INT (3),
cont>     SPECIES_NUMBER INT (3)
cont>   )
cont>     CREATE INDEX I1 ON SPECIES (FAMILY_NUMBER)
cont>     CREATE TABLE SIGHTING
cont>   ( SPECIES_NUMBER INT (3),
cont>     COMMON_NAME   CHAR (40),
cont>     CITY  CHAR (20),
cont>     STATE CHAR (20),
cont>     SIGHTING_DATE  DATE ANSI,
cont>     NOTES_NUMBER INT (5))
cont>     CREATE INDEX I2 ON SIGHTING (SPECIES_NUMBER)
cont>     CREATE TABLE FIELD_NOTES
cont>   ( WEATHER CHAR (30),
cont>     TIDE CHAR (15),
cont>     SIGHTING_TIME TIMESTAMP(2),
cont>     NOTES CHAR (500),
cont>     NOTES_NUMBER INT (5))
cont>     CREATE INDEX I3 ON FIELD_NOTES (NOTES_NUMBER)
cont> ;
SQL> DISCONNECT ALL;
SQL> ATTACH 'FILENAME birdlist';
SQL> --
SQL> -- The following CREATE STORAGE MAP statements place and
SQL> -- override thresholds on uniform storage area.
SQL> --
SQL> -- Note that the default threshold clause for the
SQL> -- storage map is not enclosed in parentheses, but each
SQL> -- threshold clause associated with a particular area is.
SQL> --
SQL> CREATE STORAGE MAP M1 FOR SPECIES
cont>     THRESHOLDS ARE (30, 50, 80)
cont>     ENABLE COMPRESSION
cont>     PLACEMENT VIA INDEX I1
cont>     STORE
cont>   IN AREA1
cont>       (THRESHOLD (10) );
SQL> --
SQL> CREATE STORAGE MAP M2 FOR SIGHTING
```

```
cont>   THRESHOLD IS (40)
cont>     STORE
cont>   RANDOMLY ACROSS (
cont>       AREA1 (THRESHOLD OF (10) ),
cont>       AREA2 (THRESHOLDS ARE (30, 50, 98) ),
cont>       AREA3
cont>   );
SQL> --
SQL> CREATE STORAGE MAP M3 FOR FIELD_NOTES
cont>   THRESHOLDS OF (50,70,90)
cont>     STORE
cont>   USING (SPECIES_NUMBER, NOTES_NUMBER)
cont>       IN AREA1
cont>           (THRESHOLDS OF (20, 80, 90) )
cont>           WITH LIMIT OF (30, 88)
cont>       IN AREA2
cont>           WITH LIMIT OF (40, 89)
cont>       IN AREA3
cont>           WITH LIMIT OF (50, 90)
cont>       OTHERWISE IN AREA4
cont>           (THRESHOLDS ARE (20, 30, 40));
SQL> --
User Storage Maps in database with filename birdlist
     M1
 For Table:             SPECIES
 Placement Via Index:   I1
 Compression is:        ENABLED
 Store clause:          STORE
     in AREA1
     (THRESHOLD (10) )
     M2
 For Table:             SIGHTING
 Compression is:        ENABLED
 Store clause:          STORE
     RANDOMLY ACROSS (
     AREA1 (THRESHOLD OF (10) ),
     AREA2 (THRESHOLDS ARE (30, 50, 98) ),
     AREA3
     )
     M3
 For Table:             FIELD_NOTES
 Compression is:        ENABLED
 Store clause:          STORE
     USING (SPECIES_NUMBER, NOTES_NUMBER)
     IN AREA1
     (THRESHOLDS OF (20, 80, 90) )
     WITH LIMIT OF (30, 88)
     IN AREA2
     WITH LIMIT OF (40, 89)
     IN AREA3
     WITH LIMIT OF (50, 90)
     OTHERWISE IN AREA4
     (THRESHOLDS ARE (20, 30, 40))
SQL> --
SQL> ROLLBACK;
```

Example 3: Creating a storage map that stores lists

This example creates a storage map that stores lists on a WORM (write-once, read-many) device in a jukebox environment. The column RESUME, which contains only lists, is stored in specially designated write-once storage areas.

```
SQL> CREATE DATABASE FILENAME test
cont>
cont> CREATE STORAGE AREA LISTS1 PAGE FORMAT IS MIXED WRITE ONCE
cont>
cont> CREATE STORAGE AREA LISTS2 PAGE FORMAT IS MIXED WRITE ONCE
cont>
cont> CREATE TABLE EMPLOYEES
cont>    (EMP_ID CHAR(5),
cont>     RESUME LIST OF VARBYTE);
SQL> --
SQL>  CREATE STORAGE MAP LISTS_MAP
cont>     STORE LISTS IN
cont>     (LISTS1,LISTS2) FOR (EMPLOYEES.RESUME)
cont>      FILL SEQUENTIALLY
cont>      IN RDB$SYSTEM;
```

Example 4: Partitioning a table both vertically and horizontally

You can divide any table both vertically and horizontally so that frequently used columns are divided among storage areas, less frequently used columns are divided among other storage areas, and rarely used columns are placed into yet another storage area.

```
SQL> -- Assume the table and storage areas have been previously defined.
SQL> --
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;
```

## CREATE TABLE Statement

Creates a temporary or persistent base table definition. A table definition consists of a list of definitions of columns that make up a row in the table.

**Persistent base tables** are tables whose metadata and data are stored in the database beyond an SQL session. The data can be shared by all users attached to the database.

**Temporary tables** are tables whose data is automatically deleted when an SQL session or module ends. The tables only materialize when you refer to them in an SQL session and the data does not persist beyond an SQL session. You can also specify whether the data is preserved or deleted at the end of a transaction within the session; the default is to delete the data. The data in temporary tables is private to the user. There are three types of temporary tables:

- Global temporary tables
- Local temporary tables
- Declared local temporary tables (see the DECLARE LOCAL TEMPORARY TABLE Statement for additional information)

The metadata for a global temporary table is stored in the database and persists beyond the SQL session. Different SQL sessions can share the same metadata. The data stored in the table cannot be shared between SQL sessions. However, the data can be shared between modules in a single SQL session. The data does not persist beyond an SQL session.

The metadata for a local temporary table is stored in the database and persists beyond the SQL session. Different SQL sessions can share the same metadata. The data stored in the table cannot be shared between different modules in a single SQL session or between SQL sessions. The data does not persist beyond an SQL session or module.

Because temporary tables are used only to hold the user’s data, which is not shared among users, no locks are needed and the data can be modified in a read-only transaction.

See the *Oracle Rdb7 Guide to Database Design and Definition* for more information on temporary tables.

When you define a table, you can also define table constraints. A **constraint** specifies a condition that restricts the values that can be stored in a table. Constraints can specify that columns contain:

- Only certain values
- Primary key values
- Unique values
- Values that cannot be null

There are two ways to specify a table definition in the CREATE TABLE statement:

- Directly by naming the table, its columns and associated data types, default values (optional), constraint definitions (optional), and formatting clauses

  You can define constraints on persistent base tables and global temporary tables only.

- OpenVMS VAX OpenVMS Alpha Indirectly by providing a path name for a repository record definition that specifies the table name, columns, and data types ♦

SQL allows you to specify the default character data type or the national character data type when defining table columns.

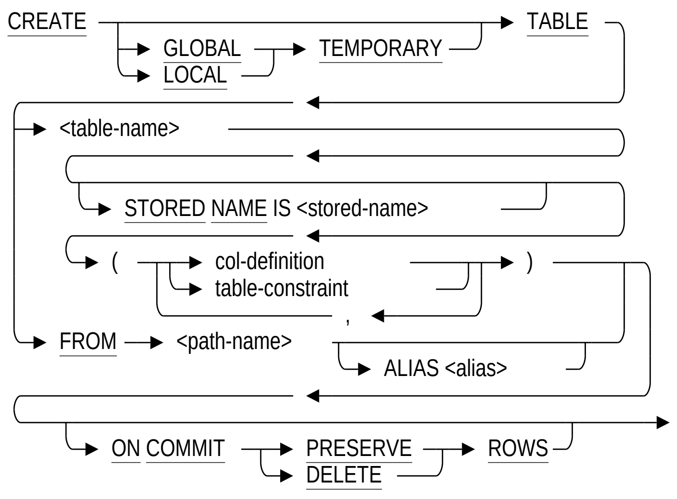
## Environment

You can use the CREATE TABLE statement:

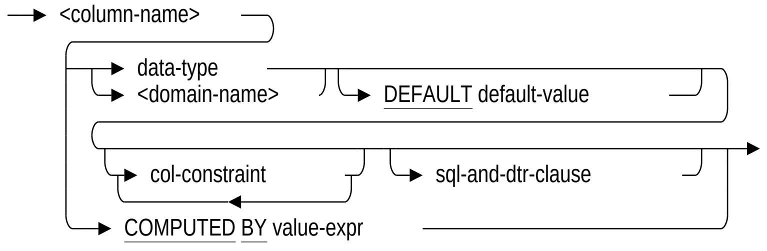
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

## CREATE TABLE Statement

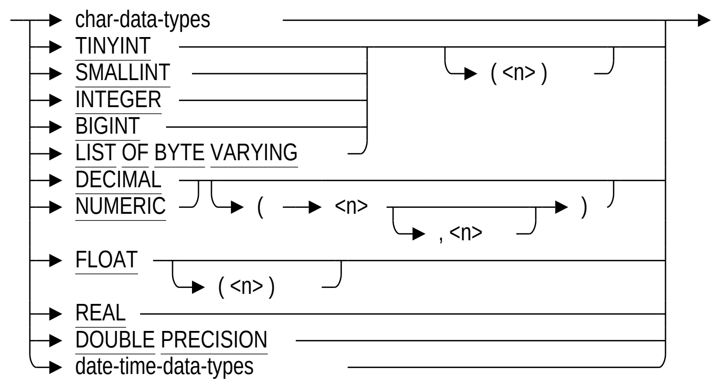
### Format



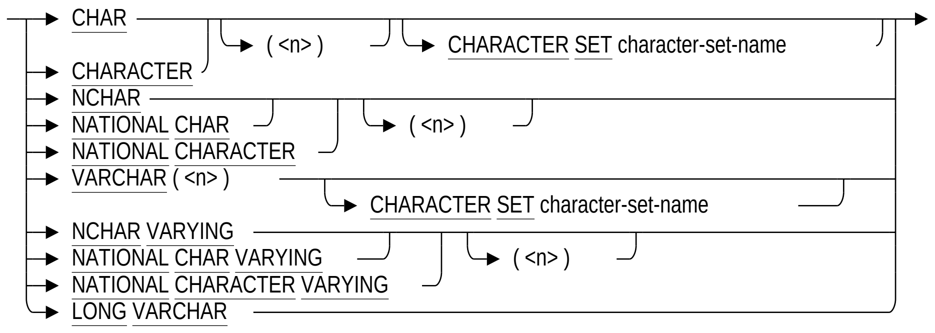
col-definition =



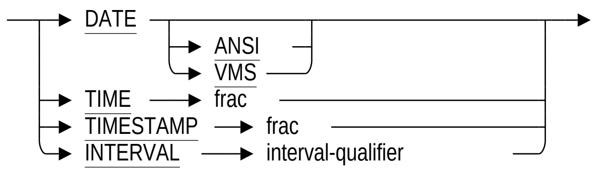
data-type =



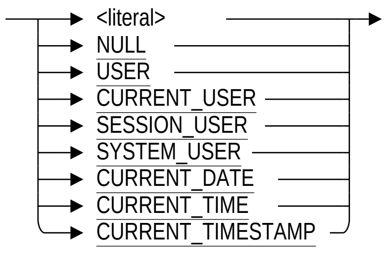
char-data-types =



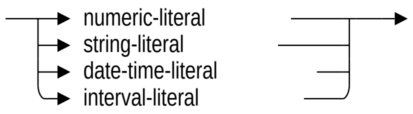
date-time-data-types =



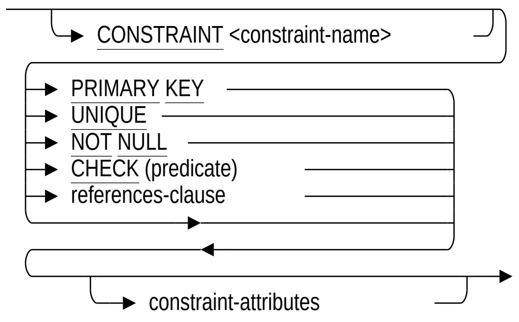
default-value =



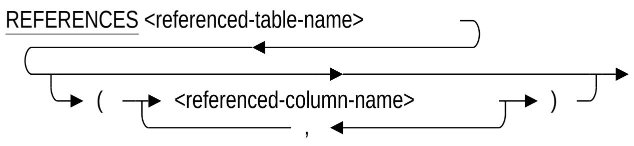
literal =



col-constraint=



references-clause =



sql-and-dtr-clause =

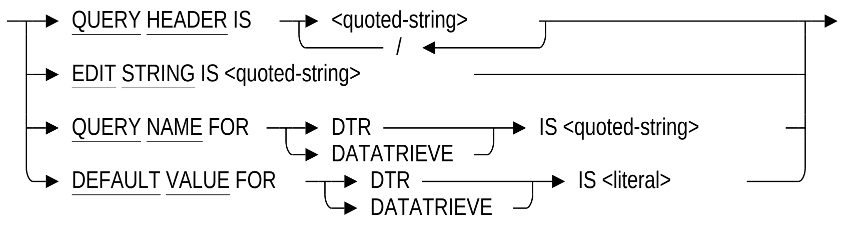


table-constraint =

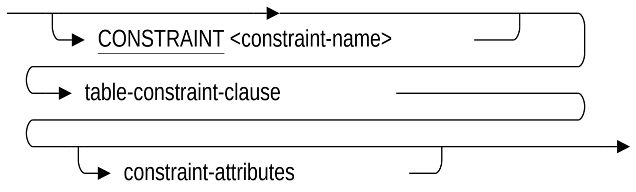
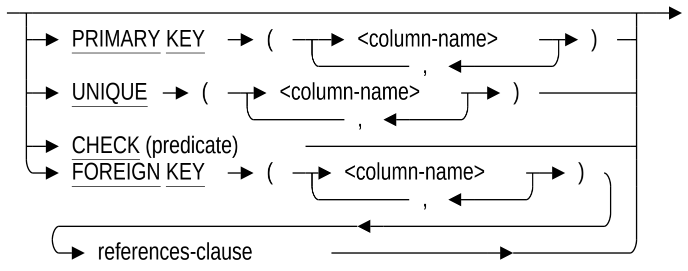
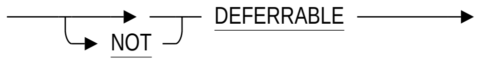


table-constraint-clause =



constraint-attributes =



## Arguments

**GLOBAL TEMPORARY**
**LOCAL**
Specifies that the table definition is either a global or local temporary table.

**table-name**
The name of the table definition you want to create. Use a name that is unique among all table and view names in the database, or in the schema if you are using a multischema database. Use any valid SQL name. (See Section 2.2 for more information on user-supplied names.)

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access a table created in a multischema database. The stored name allows you to access multischema definitions using interfaces, such as Oracle RMU, the Oracle Rdb management utility, that do not recognize multiple schemas in one database. You cannot specify a stored name for a table in a database that does not allow multiple schemas. For more details about stored names, see Section 2.2.4.

**col-definition**
The definition for a column in the table. SQL gives you two ways to specify column definitions:

- By directly specifying a data type to associate with a column name
- By naming a domain that indirectly specifies a data type to associate with a column name

Either way also allows options for specifying default values, column constraints, and formatting clauses.

**column-name**
The name of a column you want to create in the table. You need to specify a column name whether you directly specify a data type in the column definition or indirectly specify a data type by naming a domain in the column definition.

**data-type**
A valid SQL data type. Specifying an explicit data type to associate with a column is an alternative to specifying a domain name. See Section 2.3 for more information on data types.

**char-data-type**
A valid SQL character data type. See Section 2.3.1 for more information on character data types.

**character-set-name**
A valid character set name.

**date-time-data-types**
A data type that specifies a date, time, or interval. See Section 2.3.5 for more information about date-time data types.

**domain-name**
The name of a domain created in a CREATE DOMAIN statement. SQL gives the column the data type specified in the domain. For more information on domains, see the CREATE DOMAIN Statement.

For most purposes, you should specify a domain instead of an explicit data type.

- Domains ensure that all columns in multiple tables that serve the same purpose have the same data type. For example, several tables in the sample personnel database refer to the domain ID_DOM.
- A domain lets you change the data type for all columns that refer to it in one operation by changing the domain itself with an ALTER DOMAIN statement.

  For example, if you want to change the data type for the column EMPLOYEE_ID from CHAR(5) to CHAR(6), you need only alter the data type for the domain ID_DOM. You do not have to alter the data type for the column EMPLOYEE_ID in the tables DEGREES, EMPLOYEES, JOB_HISTORY, or SALARY_HISTORY, nor do you have to alter the column MANAGER_ID in the DEPARTMENTS table.

However, you might not want to use domains when you create tables if:

- Your application must be compatible with the ANSI/ISO 1989 standard. Domains are not part of the ANSI/ISO 1989 standard. (Domains are part of the ANSI/ISO SQL 1992 standard.)
- You are creating intermediate result tables that do not need the advantages of domains.

**DEFAULT default-value**
Provides a default value for a column if a row that is inserted does not include a value for that column. You can use literals, the CURRENT_DATE, CURRENT_TIME, or CURRENT_TIMESTAMP keyword, the NULL keyword, the user name, the session user name, or the system user name as default values.

If you do not specify a default value, a column inherits the default value from the domain. If you do not specify a default value for either the column or the domain, SQL assigns NULL as the default value.

You cannot specify a default value if you specify a computed column.

Remember that the default value for a column is not the same as the missing value that you can specify using the RDO interface. See the *Oracle Rdb7 Guide to Database Design and Definition* for a discussion of the difference between default value and missing value.

**default-value**
Specifies the default value of a column. The default value must be of the same data type as the column. The following table lists the valid values:

| Default Value | Description |
|---|---|
| literal | A value expression. Literal values can be numeric, character string, or date data types. |
| NULL | A null value. |
| USER | The current, active user name for a request. |
| CURRENT_USER | The current, active user name for a request. If a definer's rights request is executing, SQL returns the definer's user name. If not, SQL returns the session user name, if it exists. Otherwise, SQL returns the system user name. |
| SESSION_USER | The current, active session user name. If the session user name does not exist, SQL returns the system user name. |
| SYSTEM_USER | The user name of the process at the time of the database attach. |
| CURRENT_DATE | The DATE data type value containing year, month, and day for date "today". |
| CURRENT_TIME | The TIME data type value containing hours, minutes, and seconds for time "now". |
| CURRENT_ TIMESTAMP | The date and time currently defined in Oracle Rdb. |

**col-constraint**
A constraint that applies to values stored in the associated column.

SQL allows column constraints and table constraints. The Usage Notes summarize the differences between column constraints and table constraints. The five types of column constraints are PRIMARY KEY, UNIQUE, NOT NULL, CHECK, and FOREIGN KEY constraints. The FOREIGN KEY constraints are created with the REFERENCES clause. A constraint has the attributes of DEFERRABLE or NOT DEFERRABLE.

You can define a column constraint on persistent base tables and global temporary tables only.

**CONSTRAINT constraint-name**
Names the column constraint. See the description of the CONSTRAINT clause for table-constraints for details.

**PRIMARY KEY**
Declares this column to be a primary key. A primary key constraint defines one or more columns whose values make a row in a table different from all others. SQL requires that values in a primary key column be unique and not null; therefore, you need not specify the UNIQUE and NOT NULL column constraints for a primary key column.

You cannot specify the primary key constraint for a computed column.

**UNIQUE**
Specifies that values in the associated column must be unique. You can use either the UNIQUE or PRIMARY KEY keywords to define one or more columns as a unique key for a table.

You cannot specify the UNIQUE constraint for a computed column or for a column defined with the LIST OF BYTE VARYING data type.

**NOT NULL**
Restricts values in the column to values that are not null.

**CHECK predicate**
Specifies a predicate that column values inserted into the table must satisfy. See Section 2.7 for details on specifying predicates.

Predicates in CHECK column constraints can refer directly only to the column with which they are associated. See the Usage Notes for details.

Avoid using ANY or ALL operators or the variation of IN followed by a SELECT expression in the CHECK predicate.

**references-clause**
Specifies the name (or names) of a column (or columns) that is a primary key or a unique key in the referenced table. When the REFERENCES clause is selected as a column constraint, the column specified in the column-definition clause becomes a foreign key for the referencing table (the table being defined). When the REFERENCES clause is selected as a table constraint, the column name or column names specified in the FOREIGN KEY clause become a foreign key for the referencing table.

A computed column cannot have a REFERENCES clause.

**REFERENCES referenced-table-name**
Specifies the name of the table that contains the unique key or primary key referenced by the referencing table. To declare a constraint that refers to a unique or primary key in another table, you must have the SQL REFERENCES or CREATE privileges to the referenced table.

**referenced-column-name**
For a column constraint, the name of the column that is a unique key or primary key in the referenced table. You cannot use a computed column as a referenced column name. For a table constraint, the referenced column name is the name of the column or columns that are a unique key or primary key in the referenced table. If you omit the referenced-column-name clause, the primary key is selected by default. The number of columns and their data types must match.

**constraint-attributes**
There are two constraint attributes: DEFERRABLE and NOT DEFERRABLE.

Specifying NOT DEFERRABLE means that evaluation of the constraint must take place when the INSERT, DELETE, or UPDATE statement executes.

Specifying DEFERRABLE means that evaluation of the constraint can take place at any later time but must happen before the next COMMIT statement. You can use the SET ALL CONSTRAINTS statement to have all constraints evaluated earlier. See the SET ALL CONSTRAINTS Statement for more information.

If you are using the default SQLV40 dialect, the default constraint attribute is DEFERRABLE. When using this dialect, Oracle Rdb displays a deprecated feature message for all constraints defined without specification of one of the constraint attributes. The default constraint attribute may change in a future release. If you are using the SQL92 dialect, the default is NOT DEFERRABLE.

**sql-and-dtr-clause**
Optional SQL and DEC DATATRIEVE formatting clause. See Section 2.5 for more information about formatting clauses.

If you specify a formatting clause for a column that is based on a domain that also specifies a formatting clause, the formatting clause in the table definition overrides the one in the domain definition.

You cannot use the clauses beginning with NO, such as the NO QUERY HEADER clause, with the CREATE TABLE statement. They are valid only with the ALTER TABLE and ALTER DOMAIN statements.

SQL does not allow you to specify a default value for DATATRIEVE when you define a computed column.

**COMPUTED BY value-expr**
Specifies that the value of this column is calculated from values in other columns and constant expressions.

If your column definition refers to a column name within a value expression, that named column must already be defined within the same CREATE TABLE statement. See Section 2.6 for information on value expressions.

In a COMPUTED BY clause, the column name that you supply in your column definition must be different from the name of any other existing column in the table.

Any column that you refer to in the definition of a computed column cannot be deleted from that table unless you first delete the computed column.

SQL does not allow the following for computed columns:

- UNIQUE constraints
- REFERENCES clauses
- PRIMARY KEY constraints
- Default values
- Default values for DATATRIEVE

For example, if the FICA_RATE for an employee is 6.10 percent of the employee's starting salary and the group insurance rate is 0.7 percent, you can define FICA_RATE and GROUP_RATE fields like this:

```
SQL> CREATE TABLE payroll_detail
cont> (salary_code CHAR(1),
cont>  starting_salary SMALLINT(2),
cont>  fica_amt
cont>    COMPUTED BY (starting_salary * 0.061),
cont>  group_rate
cont>    COMPUTED BY (starting_salary * 0.007));
```

When you use this type of definition, you only have to store values in the salary_code and starting_salary columns. The FICA and group insurance deduction columns are computed automatically when the columns fica_amt or group_rate are selected.

Example 11 shows a COMPUTED BY column that uses a select expression.

**table-constraint**
A constraint definition that applies to the whole table.

SQL allows column constraints and table constraints. The Usage Notes summarize the differences between the two types of constraints. The four types of table constraints are PRIMARY KEY, UNIQUE, CHECK, and FOREIGN KEY constraints. You can also define the constraint attributes DEFERRABLE or NOT DEFERRABLE.

A column must be defined in a table before you can specify the column in a table constraint definition.

You can define a table constraint on persistent base tables and global temporary tables only.

**CONSTRAINT constraint-name**
Specifies a name for a column or table constraint. The name is used for a variety of purposes:

- The INTEG_FAIL error message specifies the name when an INSERT, UPDATE, or DELETE statement violates the constraint.
- The ALTER TABLE table-name DROP CONSTRAINT constraint-name statement specifies the name to delete a table constraint.
- The SHOW TABLE statements display the names of column and table constraints.
- The EVALUATING clause of the SET TRANSACTION and DECLARE TRANSACTION statements specifies constraint names.

The CONSTRAINT clause is optional. If you omit the constraint name, SQL creates a name. However, Oracle Rdb recommends that you always name column and table constraints. The constraint names generated by SQL may be obscure and, in programs, may change between compile time and run time. If you supply a constraint name with the CONSTRAINT clause, it must be unique in the database or in the schema if you are using a multischema database.

**PRIMARY KEY column-name**
Used to declare a column or columns as a primary key for the table being defined. Any foreign key that refers to this column must refer to this primary key. You cannot declare a computed column as a primary key.

**UNIQUE column-name**
The name of a column or columns in the table being defined that are part of a unique key. You cannot declare a computed column as a unique column name.

**CHECK (predicate)**
Specifies a predicate that column values inserted into the table must satisfy. See Section 2.7 for information on specifying predicates.

Predicates in CHECK table constraints can refer to any column in the table. Column select expressions within the predicate can refer to other tables in the database.

You cannot create views within table constraints that specify the CHECK predicate. An error is returned if you attempt to create a view in this situation.

**FOREIGN KEY column-name**
The name of a column or columns that you want to declare as a foreign key in the table you are defining (referencing table). You cannot declare a computed column as a foreign key.

**references-clause**
The REFERENCES clause is explained earlier in this Arguments list.

OpenVMS VAX OpenVMS Alpha

**FROM path-name**
Specifies the repository path name of a repository record definition. SQL creates the table using the definition from this record and gives the table the name of the record definition.

You can create a table using the FROM path-name clause only if the record definition in the repository was originally created using the repository Common Dictionary Operator (CDO) utility. For instance, you cannot create a table using the FROM path-name clause if the record definition was created in the repository as part of an SQL session.

If the repository record contains a nested record definition, you cannot create a table based on it.

Creating a table based on a repository record definition is useful when many applications share the same definition. Changes to the common definition can be automatically reflected in all applications that use it.

**Note**

Changes by other users or applications to the record definition in the repository affect the table definition once the database is integrated to match the repository with an INTEGRATE DATABASE . . . ALTER FILES statement. If those changes include deleting records or fields on which tables or table columns are based, any data in the dependent table or table column is lost after the next INTEGRATE DATABASE . . . ALTER FILES statement executes.

---

You can use the FROM clause only if the database was attached specifying PATHNAME. You can specify either a full repository path name or a relative repository path name.

You cannot define constraints or any other table definition clauses, such as DATATRIEVE formatting clauses, when you use the FROM path-name form of the CREATE TABLE statement. This restriction does not prevent you from using an ALTER TABLE statement to add them later.

You cannot use the FROM path-name clause when embedding a CREATE TABLE statement within a CREATE DATABASE statement.

The FROM clause is available only on OpenVMS platforms. ♦

OpenVMS VAX OpenVMS Alpha

**ALIAS alias**

Specifies a name for an attach to a particular database. SQL adds the table definition to the database referred to by the alias.

If you do not specify an alias, SQL adds the table definition to the default database. See Section 2.2.2 for more information on default databases and aliases.

The ALIAS clause is available only on OpenVMS platforms. ♦

**ON COMMIT PRESERVE ROWS**
**ON COMMIT DELETE ROWS**

Specifies whether data is preserved or deleted after a COMMIT statement for global or local temporary tables only.

The default, if not specified, is ON COMMIT DELETE ROWS.

## Usage Notes

- When the CREATE TABLE statement executes, SQL adds the table definition to the database.

  OpenVMS VAX OpenVMS Alpha

  If you declared the database with the PATHNAME specification, the definition is also added to the repository. ♦

OpenVMS VAX OpenVMS Alpha

- The CREATE TABLE statement fails when both of the following are true:
  - The database to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The database was attached using the FILENAME argument.

  Under these circumstances, the statement fails with the following error when you issue it:

  ```
  %RDB-E-NO_META_UPDATE, metadata update failed
  -RDMS-F-CDDISREQ, CDD required for metadata updates
                    is not being maintained
  ```
  ♦

OpenVMS VAX OpenVMS Alpha

- It is possible when using the repository to define record structures that are not acceptable to Oracle Rdb.

  The repository is intended as a generic data repository that can hold data structures available to many layered products and languages.

  These data structures may not always be valid when applied to the relational data model used by Oracle Rdb.

  The following are some of the common incompatibilites between the data structures of the repository and Oracle Rdb.
  - %CDD-E-PRSMISSNG, attribute value is missing

    This error can occur when a record definition in the repository contains a VARIANTS clause.
  - %CDD-E-INVALID_RDB_DTY, data type of field is not supported by Oracle Rdb

    This error can occur when a record definition in the repository contains an OCCURS clause.
  - %CDD-E-DTYPE_REQUIRED, field must have a data type for inclusion in an Oracle Rdb database

    This error can occur when a record definition in the repository contains another nested record definition. Oracle Rdb can only accept field definitions in a record definition.
  - %CDD-E-INVALID_RDB_DIM, record PARTS has dimension and cannot be used by Oracle Rdb

    This error occurs when a record definition in the repository contains an ARRAY clause. ♦

- The CREATE TABLE statement creates a default access privilege set for the table that gives the creator all privileges to the database and all other users no privileges. This means that new tables have a PUBLIC access of NONE.

  To override default PUBLIC access for newly created tables, define an identifier with the name DEFAULT in the system privileges table. The access rights that you give to this identifier on your database will then be assigned to any new tables that you create.

  1. Assigning the SELECT and UPDATE privileges to the database with alias TEST1

     ```
     SQL> ATTACH 'ALIAS test1 FILENAME mf_personnel';
     SQL> SHOW PROTECTION ON DATABASE test1;
     Protection on Alias TEST1
         (IDENTIFIER=[dbs,smallwood],ACCESS=SELECT+INSERT+UPDATE+DELETE+
           SHOW+CREATE+ALTER+DROP+DBCTRL+OPERATOR+DBADM+SECURITY+DISTRIBTRAN)
         (IDENTIFIER=[*,*],ACCESS=NONE)
     SQL> GRANT SELECT, UPDATE ON DATABASE ALIAS TEST1
     cont> TO DEFAULT;
     ```

  2. Committing and disconnecting the transaction to make the change in protection occur

     ```
     SQL> COMMIT;
     SQL> DISCONNECT ALL;
     ```

  3. Receiving all access rights to the new table TABLE1

     The protection on existing tables in the database is not changed; however, any new tables that you define receive the protection specified by the DEFAULT identifier. In this example, the owner (SMALLWOOD) receives all the access rights to the new table TABLE1, and all other users receive the SELECT and UPDATE access rights specified by the DEFAULT identifier.

     ```
     SQL> ATTACH 'ALIAS test1 FILENAME mf_personnel';
     SQL> SET TRANSACTION READ WRITE;
     SQL> CREATE TABLE test1.table1
     cont>       (last_name_dom CHAR(5),
     cont>        year_dom SMALLINT);
     SQL> SHOW PROTECTION ON test1.table1;
     Protection on Table TEST1.TABLE1
         (IDENTIFIER=[dbs,smallwood],ACCESS=SELECT+INSERT+UPDATE+DELETE+
           SHOW+CREATE+ALTER+DROP+DBCTRL+REFERENCES)
         (IDENTIFIER=[*,*],ACCESS=SELECT+UPDATE)
     ```

The DEFAULT identifier is typically present on an OpenVMS system because the DEFAULT OpenVMS account is always present and cannot be removed. However, it is possible to remove the DEFAULT identifier associated with that account. If the DEFAULT identifier has been removed from your system, Oracle Rdb returns an error message.

```
SQL> GRANT INSERT ON DATABASE ALIAS TEST1 to DEFAULT;
%SYSTEM-F-NOSUCHID, unknown rights identifier
```

- You must execute the CREATE TABLE statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with characteristics specified in the most recent DECLARE TRANSACTION statement.
- You cannot execute the CREATE TABLE statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on the RDB$SYSTEM storage area.
- You should consider what value, if any, you want to use for the default value for a column. You can use a value such as NULL or Not Applicable that clearly demonstrates that no data was inserted into a column. If a column usually contains a particular value, you can use that value as the default. For example, if most company employees work full-time, you could make full-time the default value for a work status column.
- If you specify a default value for a column that you base on a domain and you have specified a default value for that domain, the default value for the column overrides the default value for the domain.
- The resulting data type for the USER, CURRENT_USER, SESSION_USER, and SYSTEM_USER keywords is CHAR(31).
- Table-specific constraints can be declared at the table level or the column level or both. These constraints can specify that columns contain only certain values, primary key values, unique values, or that values cannot be missing (null). Multiple constraints can be declared at both the table and column level.

  On both levels, you can specify definitions of unique, primary, and foreign keys, and foreign key references to unique or primary keys. You can also specify constraint evaluation time (either commit or update).

  On the table level, you can define constraints for multicolumn keys.

  On the column level, you can restrict the values of columns to values that are not null (missing).

- You can control when the database system evaluates constraints using the SET ALL CONSTRAINTS statement.
- If you defined constraints as NOT DEFERRABLE, they must be evaluated when the INSERT, DELETE, or UPDATE statement executes. You cannot use either the SET ALL CONSTRAINTS statement or the SET TRANSACTION EVALUATING statement to change the evaluation time.
- Constraints specify a condition that restricts the values that can be stored in a table. The INSERT, UPDATE, or DELETE statements that violate the condition fail. The database system generates an RDB$_INTEG_FAIL error, and SQL returns an SQLCODE value of –1001.

  You can control when the database system evaluates constraints in the EVALUATING clause of DECLARE and SET TRANSACTION statements. By default, all constraints are evaluated when a transaction issues a COMMIT statement. However, if you specify VERB TIME for specific constraints in the EVALUATING clause of a DECLARE or SET TRANSACTION statement, the database system evaluates those constraints whenever UPDATE, INSERT, or DELETE statements execute. SQL allows column constraints and table constraints. The semantics and syntax for the two types of constraints are similar, but not identical. The following list summarizes the differences:

  – Column constraints allow the UNIQUE argument; table constraints allow the UNIQUE (column-name) argument. Specifying UNIQUE for a series of column definitions is more restrictive than specifying UNIQUE and a list of the same columns because SQL requires only that the combination of columns in a UNIQUE (column-name) table constraint be unique.

```
SQL> CREATE TABLE TEMP1
cont> ( COL1 REAL NOT NULL UNIQUE CONSTRAINT C1,
cont>   COL2 REAL NOT NULL UNIQUE CONSTRAINT C2,
cont>   COL3 REAL NOT NULL UNIQUE CONSTRAINT C3 );
SQL>
SQL> CREATE TABLE TEMP2
cont> ( COL4 REAL NOT NULL CONSTRAINT C4,
cont>   COL5 REAL NOT NULL CONSTRAINT C5,
cont>   COL6 REAL NOT NULL CONSTRAINT C6,
cont>   UNIQUE (COL4, COL5, COL6) CONSTRAINT C7  );
SQL>
SQL> INSERT INTO TEMP1 VALUES (1,1,1);
1 row inserted
SQL> INSERT INTO TEMP2 VALUES (1,1,1);
1 row inserted
SQL> COMMIT;
SQL>
SQL> -- This fails because the values
SQL> -- in COL1 will not be unique:
SQL> INSERT INTO TEMP1 VALUES (1,2,2);
1 row inserted
SQL> COMMIT;
%RDB-E-INTEG_FAIL, violation of constraint C1 caused operation to fail
SQL>
SQL> ROLLBACK;
SQL>
SQL> -- This succeeds because the *combination*
SQL> -- of the columns is still unique:
SQL> INSERT INTO TEMP2 VALUES (1,2,2);
1 row inserted
SQL> COMMIT;
```

– The CHECK constraints have the same syntax for column constraints as for table constraints. The only syntactic distinction between the two CHECK constraints is that CHECK table constraints are separated from column definitions by commas, and CHECK column constraints are not.

  The predicate in a CHECK column constraint can refer directly only to the column with which it is associated. The predicate in a CHECK table constraint can refer directly to any column in the table. Either type of CHECK constraint, however, can refer to columns in other tables in the database through column select expressions in the predicate.

  The predicate of a CHECK constraint must not be false. It may be unknown. The constraint COL 10 > 100 would allow values 101, 1000, and NULL. It would not allow the value 99.

```
SQL> -- Cannot directly refer to TEST1 in
SQL> -- column constraint for TEST2:
SQL> CREATE TABLE TEST
cont> ( TEST1 CHAR(5),
cont>   TEST2 CHAR(5)
cont>   CHECK (TEST2 <> TEST1)
cont> );
%SQL-F-COLNOTVAL, The column CHECK constraint cannot refer to the
column TEST1
SQL> -- To get around the problem, make the CHECK constraint a table
SQL> -- constraint by separating it from the column with a comma:
SQL> CREATE TABLE TEST
cont> ( TEST1 CHAR(5),
cont>   TEST2 CHAR(5),
cont>   CHECK (TEST2 <> TEST1)
cont> );
SQL> COMMIT;
SQL> INSERT INTO TEST VALUES ('1','1');
1 row inserted
SQL> COMMIT;
%RDB-E-INTEG_FAIL, violation of constraint TEST_CHECK1 caused operation
to fail
SQL> ROLLBACK;
SQL> -- This table shows that a CHECK column constraint
SQL> -- can refer to other tables in column select expressions:
SQL> CREATE TABLE TEST0
cont> ( TEST1 CHAR(5),
cont>   TEST2 CHAR(5)
cont>   CHECK (TEST2 NOT IN
cont>   (SELECT TEST1 FROM TEST0) )
cont> );
```

- An alternative to specifying unique column or table constraints is to use CREATE INDEX statements with the UNIQUE keyword. Specifying UNIQUE indexes generally gives better performance than specifying logically equivalent constraints in a table definition. However, you cannot specify UNIQUE indexes for references-clause constraints that refer to other tables.
- The REFERENCES clause can declare the one or more corresponding columns in the referenced table that comprise a unique or primary key. If not, the referenced table must include a PRIMARY KEY constraint at the table level specifying the corresponding column or columns.
- The values in a foreign key must match the values in the related unique key or primary key. SQL considers that the foreign key matches the related unique key or primary key when either of the following statements is true:
  - A column in the foreign key contains a null value. In this case, the foreign key is null. SQL considers that a null foreign key matches the related unique key or primary key.
  - None of the columns in the foreign key contains a null value, and the set of values in the foreign key also exists in the unique key or primary key. In other words, the foreign key matches the related unique key or primary key when for every row in the referencing table, there is a row in the referenced table where the corresponding columns are equal.

  The following example illustrates the first type of match. The null value stored in column B2 of the table FOREIGN makes the foreign key of B1 and B2 a null foreign key. As a null foreign key, B1 and B2 match the primary key A1 and A2 in the table PRIMARY.

```
SQL> CREATE TABLE PRIMARY_TAB
cont> ( A1 INTEGER,
cont>   A2 INTEGER,
cont>   PRIMARY KEY (A1, A2),
cont>   A3 INTEGER);
SQL>
SQL> INSERT INTO PRIMARY_TAB (A1, A2, A3)
cont>               VALUES (1, 1, 1);
1 row inserted
SQL>
SQL> CREATE TABLE FOREIGN_TAB
cont> ( B1 INTEGER,
cont>   B2 INTEGER,
cont>   FOREIGN KEY (B1, B2)
cont>     REFERENCES PRIMARY_TAB (A1, A2),
cont>   B3 CHAR(5));
SQL> -- The following command stores a null value in column B2:
SQL> INSERT INTO FOREIGN_TAB (B1, B3) VALUES (2, 'AAAAA');
1 row inserted
```

This example shows the second type of match. The values stored in columns D1 and D2 (the foreign key) of the table FOREIGN_2 exactly match the values stored in columns C1 and C2 (the primary key) of the table PRIMARY_2.

```
SQL> CREATE TABLE PRIMARY_2
cont> (C1 INTEGER,
cont>   C2 INTEGER,
cont>   PRIMARY KEY (C1, C2),
cont>   C3 INTEGER);
SQL>
SQL> INSERT INTO PRIMARY_2 (C1, C2, C3)
cont>             VALUES (5, 3, 2);
1 row inserted
SQL>
SQL> CREATE TABLE FOREIGN_2
cont> ( D1 INTEGER,
cont>   D2 INTEGER,
cont>   FOREIGN KEY (D1, D2)
cont>     REFERENCES PRIMARY_2 (C1, C2),
cont>   D3 CHAR(5));
SQL> --
SQL> INSERT INTO FOREIGN_2 (D1, D2, D3) VALUES (5, 3, 'BBBBB');
1 row inserted
```

- You can use table-specific constraints to:
  - Maintain referential integrity by establishing a clear, visible set of rules
  - Attach the desired integrity rules directly to the definition of a table

  - Avoid defining multiple, seemingly independent constraints to accomplish the same task

- Constraints should not specify columns defined as segmented strings, as only the segmented string ID is referenced, not the actual segmented string.
- Within the table definition, constraints can apply to the values in specific rows of a table, to the entire contents of a table, or to states existing between multiple tables.
- Within the table definition, Oracle Rdb first defines new versions of columns. Then, SQL defines constraints and evaluates them. Therefore, if columns and constraints are defined within the same table definition, constraints can use any of the columns defined in this table before or after the constraint definition text.
- Adding a constraint to the database causes the constraint criteria to be evaluated if there is data to validate. If existing data does not meet the criteria, an exception is produced and the definition fails.
- If table-specific constraints are declared with a CREATE TABLE statement and the definition of the generated constraint fails, the definition of the table also fails.
- The CREATE TABLE statement adds the table definition and any associated constraint definitions to the physical database.

  OpenVMS VAX OpenVMS Alpha If the database was attached with the PATHNAME argument, the definitions are stored in the repository, ensuring consistency between the database definitions and the repository definitions. ♦
- To ensure that you do not define redundant, table-specific constraints, you should display all constraints and triggers for the affected table using the SHOW TABLE statement.
- If a constraint fails at commit time, the update operation must be manually rolled back.
- You can create up to 8191 tables. This value is an architectural limit restricted by the on-disk structure. When you exceed the maximum limit, Oracle Rdb issues an error message.

  If you delete older tables, Oracle Rdb recycles their identifiers so that CREATE TABLE statements can succeed even after the maximum is reached.
- CREATE TABLE statements in programs must precede (in the source file) all other data definition language (DDL) statements that refer to the table.

- You can specify the national character data type by using the NCHAR, NATIONAL CHAR, NCHAR VARYING, or NATIONAL CHAR VARYING data types. The national character data type is defined by the database national character set when the database is created. See Section 2.3 for more information regarding national character data types.
- You can specify the length of the data type in characters or octets. By default, data types are specified in octets. By preceding the CREATE TABLE command with the SET CHARACTER LENGTH 'CHARACTERS' statement, SET DIALECT 'MIA' statement, SET DIALECT 'SQL89' statement, or SET DIALECT 'SQL92' statement, you change the length to characters. For more information, see the SET CHARACTER LENGTH Statement and the SET DIALECT Statement, respectively.
- You can create a table without specifying a character set for the column that defines the table with the database default character set.
- You can create a table specifying a character set for the column other than the database default or national character sets.
- Because data in temporary tables is private to a session, you cannot use temporary tables in as many places as you use persistent base tables. In particular, note the following points when you use temporary tables:
  - Temporary tables are stored in virtual memory, not in a storage area. They use the same storage segment layout as persistent base tables, but they use additional space in memory for management overhead. On OpenVMS, temporary tables use 56 bytes per row for management overhead; on Digital UNIX, they use 88 bytes.

    See the *Oracle Rdb7 Guide to Database Design and Definition* for information on estimating the virtual memory needs of temporary tables.
  - You cannot alter a temporary table. To alter a global or local temporary table, you must delete the table and create it again.
  - You can truncate global temporary tables using the TRUNCATE TABLE statement. You cannot truncate local temporary tables.
  - Global and local temporary tables cannot contain data of the data type LIST OF BYTE VARYING.
  - You can define column and table constraints for global temporary tables, but not for local temporary tables. The columns in both global and local temporary tables can reference domain constraints.

Constraints on a global temporary table can only refer to another global temporary table. However, if the referenced target table specifies ON COMMIT DELETE ROWS, the source table must also specify ON COMMIT DELETE ROWS. This restriction does not apply when the referenced target table specifies ON COMMIT PRESERVE ROWS.

  - A storage map can refer to a global or local temporary table, but the storage map can only specify whether to enable or disable compression on the table. You can enable or disable compression on a temporary table only prior to inserting any data into the table. Compression is enabled by default.
  - You can use triggers with global temporary tables only.
  - You cannot define indexes for global or local temporary tables.
  - You cannot specify a global or local temporary table in the RESERVING clause of a SET TRANSACTION statement.
  - Oracle Rdb does not journal changes to global or local temporary tables.

- The following are allowed with global or local temporary tables:
  - You can delete temporary tables using the DROP TABLE statement.
  - A view can refer to a temporary table.
  - You can use dbkeys with temporary tables.
  - You can grant and revoke privileges only using the ALL keyword.
  - You can write to a temporary table during a read-only transaction.
- Table 6–4 summarizes the actions you can take with temporary tables and when you can refer to temporary tables.

**Table 6–4 Using Temporary Tables**

| | Types of Temporary Tables | | |
|---|---|---|---|
| **Action** | **Global** | **Local** | **Declared Local** |
| Delete table | Yes | Yes | No |
| Modify table | No | No | No |
| Truncate table | Yes | No | No |

(continued on next page)

**Table 6–4 (Cont.) Using Temporary Tables**

| | Types of Temporary Tables | | |
|---|---|---|---|
| **Action** | **Global** | **Local** | **Declared Local** |
| Define constraints on table or column | Yes | No | No |
| Refer to table in constraint definition | Yes[2] | Yes | No |
| Refer to domain constraints | Yes | Yes | Yes |
| Refer to table in storage map | Yes | Yes | No |
| Refer to table in view | Yes | Yes | No |
| Grant privileges on temporary table | Yes | Yes | No |
| Refer to table in outline | Yes | Yes | No[1] |
| Define indexes on table | No | No | No |
| Use dbkeys on table | Yes | Yes | Yes |
| Use triggers with table | Yes | No | No |
| Refer to table in COMMENT ON statement | Yes | Yes | No |
| Contain LIST OF BYTE VARYING data | No | No | No |
| Specify in RESERVING clause | No | No | No |
| Write to table during read-only transaction | Yes | Yes | Yes |
| Create in a read-only transaction | No | No | Yes |
| Refer to a table in a computed by column | Yes | Yes | No |

[1]You can refer to a declared local temporary table if it is defined inside a stored module.

[2]From a temporary table only.

For information about declared local temporary tables, see the DECLARE LOCAL TEMPORARY TABLE Statement.

- When deleting and creating temporary tables using the same table name, you must commit the delete operation before starting the create operation. For example:

```
SQL> CREATE GLOBAL TEMPORARY TABLE t (c INTEGER);
SQL> INSERT INTO t (c) VALUES (2);
1 row inserted
SQL> COMMIT;
SQL> DROP TABLE t;
SQL> COMMIT;
SQL> CREATE GLOBAL TEMPORARY TABLE t (c INTEGER);
```

If you do not commit after the delete operation, you receive the following error message:

```
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-E-INVTEMPTBL, invalid use of temporary table
-RDMS-F-RELEXTS, there is another relation named T in this database
```

## Examples

Example 1: Creating new tables with primary and foreign keys

In this example, the CREATE TABLE statement is used to create the EMPLOYEES_2, SALARY_HISTORY_2, and WORK_STATUS_2 tables in the personnel database. It specifies column definitions based on domain definitions for the entire database.

The FOREIGN KEY constraint specified in the SALARY_HISTORY_2 table must match the PRIMARY KEY constraint specified in the EMPLOYEES_2 table.

Note also that the CHECK constraint specified is a table constraint because it is separated by commas from the column to which it refers. In this case, a column constraint on EMPLOYEE_ID would have the same effect because it refers only to the single column EMPLOYEE_ID.

Because the dialect is SQL92, the default for constraint evaluation time is NOT DEFERRABLE.

```
SQL> -- *** Set Dialect ***
SQL> --
SQL> SET DIALECT 'SQL92';
SQL> --
SQL> -- *** Create tables ***
SQL> --
SQL> CREATE TABLE WORK_STATUS_2
cont>   (
cont>   STATUS_CODE      STATUS_CODE_DOM
cont>     CONSTRAINT WS2_STATUS_CODE_PRIMARY
cont>     PRIMARY KEY,
cont>   STATUS_NAME      STATUS_NAME_DOM,
cont>   STATUS_TYPE      STATUS_DESC_DOM
cont>   );
SQL> --
SQL> CREATE TABLE EMPLOYEES_2
cont>   (
cont>   EMPLOYEE_ID      ID_DOM
cont>     CONSTRAINT E2_EMPLOYEE_ID_PRIMARY
cont>     PRIMARY KEY,
cont>   LAST_NAME        LAST_NAME_DOM,
cont>   FIRST_NAME       FIRST_NAME_DOM,
cont>   MIDDLE_INITIAL   MIDDLE_INITIAL_DOM,
cont>   ADDRESS_DATA_1   ADDRESS_DATA_1_DOM,
cont>   ADDRESS_DATA_2   ADDRESS_DATA_2_DOM,
cont>   CITY             CITY_DOM,
cont>   STATE            STATE_DOM,
cont>   POSTAL_CODE      POSTAL_CODE_DOM,
cont>   SEX              SEX_DOM
cont>     CONSTRAINT     EMPLOYEE_SEX_VALUES
cont>     CHECK          (
cont>                    SEX IN ('M', 'F') OR SEX IS NULL
cont>                    ),
cont>   BIRTHDAY         DATE_DOM,
cont>   STATUS_CODE      STATUS_CODE_DOM
cont>     CONSTRAINT E2_STATUS_CODE_FOREIGN
cont>     REFERENCES WORK_STATUS_2 (STATUS_CODE),
cont>     CONSTRAINT     EMP_STATUS_CODE_VALUES_2
cont>     CHECK          (
cont>                    STATUS_CODE IN ('0', '1', '2')
cont>                    OR STATUS_CODE IS NULL
cont>                    )
cont>     );
```

```
SQL> --
SQL> CREATE TABLE SALARY_HISTORY_2
cont>   (
cont>   EMPLOYEE_ID      ID_DOM
cont>     CONSTRAINT SH2_EMPLOYEES_ID_FOREIGN
cont>     REFERENCES EMPLOYEES_2 (EMPLOYEE_ID),
cont>   SALARY_AMOUNT    SALARY_DOM,
cont>   SALARY_START     DATE_DOM,
cont>   SALARY_END       DATE_DOM
cont>   );
SQL>
```

Example 2: Creating a table with many SQL data types

The following example is an excerpt from the sample program sql_all_datatypes created during installation of Oracle Rdb in the Samples directory. For a variety of languages, sql_all_datatypes illustrates how you declare program variables to match a variety of data types, and how you can specify those variables in SQL statements when you store and retrieve column values or null values.

This example shows the CREATE TABLE statement from the sql_all_datatypes program.

```
EXEC SQL CREATE TABLE ALL_DATATYPES_TABLE
        (
        CHAR_COL                CHAR(10),
        SMALLINT_COL            SMALLINT,
        SMALLINT_SCALED_COL     SMALLINT (3),
        INTEGER_COL             INTEGER,
        INTEGER_SCALED_COL      INTEGER (2),
        QUADWORD_COL            QUADWORD,
        QUADWORD_SCALED_COL     QUADWORD (5),
        REAL_COL                REAL,
        DOUBLE_PREC_COL         DOUBLE PRECISION,
        DATE_COL                DATE,
        VARCHAR_COL             VARCHAR(40)
        );
```

Example 3: Specifying default values for columns

The following example illustrates the use of default values for columns. Each salesperson enters his or her own daily sales information into the DAILY_SALES table.

```
SQL> --
SQL> CREATE TABLE DAILY_SALES
cont> --
cont> -- The column SALESPERSON is based on LAST_NAME_DOM and
cont> -- the default value is the user name of the person who
cont> -- enters the information:
cont> (SALESPERSON LAST_NAME_DOM DEFAULT USER,
cont> --
cont> -- Typical work day is 8 hours:
cont>   HOURS_WORKED SMALLINT DEFAULT 8,
cont>   HOURS_OVERTIME SMALLINT,
cont>   GROSS_SALES INTEGER );
SQL> --
SQL> -- Insert daily sales information accepting the
SQL> -- default values for SALESPERSON and HOURS_WORKED:
SQL> --
SQL> INSERT INTO DAILY_SALES
cont> (HOURS_OVERTIME, GROSS_SALES )
cont> VALUES
cont> (1, 2499.00);
1 row inserted
SQL> SELECT * FROM DAILY_SALES;
 SALESPERSON      HOURS_WORKED   HOURS_OVERTIME   GROSS_SALES
 KILPATRICK                  8                1          2499
1 row selected
```

Example 4: Violating a constraint indirectly with the DELETE statement

Constraints prevent INSERT statements from adding rows to a table that do not satisfy conditions specified in the constraint. Constraints also prevent DELETE or UPDATE statements from deleting or changing values in a table if the deletion or change violates the constraint on another table in the database. The following example illustrates that point:

```
SQL> -- TEST has no constraints defined for it, but it is subject to
SQL> -- restrictions nonetheless because of the constraint specified
SQL> -- in TEST2:
SQL> CREATE TABLE TEST
cont> (COL1 REAL);
SQL>
SQL> CREATE TABLE TEST2
cont> (COL1 REAL,
cont> CHECK (COL1 IN
cont>   (SELECT COL1 FROM TEST))
cont> );
SQL> COMMIT;
SQL>
SQL> INSERT INTO TEST VALUES (1);
1 row inserted
SQL> INSERT INTO TEST2 VALUES (1);
1 row inserted
SQL> COMMIT;
SQL> -- This DELETE statement will fail because it will cause COL1 in
SQL> -- TEST2 to contain a value without the same value in COL1 of TEST:
SQL> DELETE FROM TEST WHERE COL1 = 1;
1 row deleted
SQL> COMMIT;
%RDB-E-INTEG_FAIL, violation of constraint TEST2_CHECK1 caused operation to
fail
```

Example 5: Evaluating constraints at verb time

By default, constraints are not evaluated until a transaction issues a COMMIT statement. You can specify that constraints be evaluated more frequently with the EVALUATING clause of the SET TRANSACTION statement.

```
SQL> CREATE TABLE TEST
cont> ( COL1 REAL,
cont>   COL2 REAL NOT NULL UNIQUE
cont>           CONSTRAINT C2   );
SQL> COMMIT;
SQL> -- This INSERT statement violates the NOT NULL UNIQUE constraint
SQL> -- on COL2, but you do not find out until the transaction issues
SQL> -- a COMMIT statement:
SQL> INSERT INTO TEST (COL1) VALUES (3);
1 row inserted
SQL> COMMIT;
%RDB-E-INTEG_FAIL, violation of constraint C2 caused operation to fail
SQL> ROLLBACK;
SQL>
SQL> -- To generate constraint errors when offending statements are issued,
SQL> -- use the EVALUATING clause in the SET TRANSACTION statement:
SQL> --
SQL> SET TRANSACTION EVALUATING C2 AT VERB TIME;
SQL> INSERT INTO TEST (COL1) VALUES (3);
%RDB-E-INTEG_FAIL, violation of constraint C2 caused operation to fail
```

Example 6: Specifying the DECIMAL data type in the CREATE TABLE statement

SQL does not support a packed decimal or numeric string data type. If you specify the DECIMAL or NUMERIC data type for a column in a CREATE TABLE or ALTER TABLE statement, SQL generates a warning message and creates the column with a data type that depends on the precision argument specified (see Section 2.3.2 for details). This example shows a CREATE TABLE statement that specifies a DECIMAL data type.

```
SQL> CREATE TABLE TEMP
cont>   (DECIMAL_EX DECIMAL);
%SQL-I-NO_DECIMAL, DECIMAL_EX is being converted from DECIMAL to INTEGER.
SQL>
```

OpenVMS VAX OpenVMS Alpha

Example 7: Basing a table on a repository record definition

In the following example, the FROM clause is used in a CREATE TABLE statement to create a table with constraints based on a repository record definition. The PARTS record (table) has a primary key based on the field (column) PART_ID and a unique key based on the field (column) PART_NO, as well as other constraints.

This example assumes that OTHER_PARTS record and OTHER_PARTS_ID field have been previously defined in the repository. It begins with defining the fields and the record in the repository using the Common Dictionary Operator utility.

```
$ !
$ ! Define CDD$DEFAULT:
$ !
$ DEFINE CDD$DEFAULT SYS$COMMON:[REPOSITORY]TABLE_TEST
$ !
$ ! Enter the respository to create new field and record definitions:
$ !
$ REPOSITORY
Welcome to CDO V2.3
The CDD/Repository V5.3 User Interface
Type HELP for help
CDO> !
CDO> ! Create the field definitions for the PARTS record:
CDO> !
CDO> DEFINE FIELD PART_NO DATATYPE IS SIGNED WORD.
CDO> DEFINE FIELD PART_ID DATATYPE IS SIGNED LONGWORD.
CDO> DEFINE FIELD PART_ID_USED_IN DATATYPE IS SIGNED LONGWORD.
CDO> DEFINE FIELD PART_QUANT DATATYPE IS SIGNED WORD.
CDO> !
CDO> ! Create the PARTS record definition by first defining the constraints
CDO> ! and then including the field definitions just created.  Note that
CDO> ! CDO creates the constraints as not deferrable.
CDO> !
CDO> DEFINE RECORD PARTS
cont>    CONSTRAINT PARTS_PMK PRIMARY KEY PART_ID
cont>    CONSTRAINT PARTS_UNQ UNIQUE PART_NO
cont>    CONSTRAINT PART_CST CHECK
cont>        (ANY P IN PARTS WITH (PART_ID IN
cont>        PARTS = PART_ID_USED_IN IN P))
cont>    CONSTRAINT PART_FRK
cont>        FOREIGN KEY PART_ID REFERENCES OTHER_PARTS OTHER_PART_ID.
cont>    PART_NO.
cont>    PART_ID.
cont>    PART_ID_USED_IN.
cont>    PART_QUANT.
cont>    END.
CDO>  !
CDO>  ! Display the RECORD PARTS:
CDO>  !
CDO>  SHOW RECORD PARTS/FULL
Definition of record PARTS
|   Contains field          PART_NO
|   |   Datatype                signed word
|   Contains field          PART_ID
|   |   Datatype                signed longword
|   Contains field          PART_ID_USED_IN
|   |   Datatype                signed longword
|   Contains field          PART_QUANT
|   |   Datatype                signed word
|   Constraint PARTS_PMK  primary key PART_ID NOT DEFERRABLE
|   Constraint PARTS_UNQ  unique PART_NO NOT DEFERRABLE
|   Constraint PART_CST (ANY (P IN PARTS WITH
|          (PART_ID IN PARTS EQ PART_ID_USED_IN IN P))) NOT DEFERRABLE
|   Constraint PART_FRK  foreign key PART_ID references OTHER_PARTS
|           OTHER_PART_ID NOT DEFERRABLE
CDO> EXIT
$ !
$ ! Entering SQL:
$ SQL
SQL> !
SQL> ! Attach to the AUTO database:
SQL> !
SQL> ATTACH 'ALIAS AUTO PATHNAME AUTO';
SQL> !
SQL> ! Create a table called PARTS using the PARTS record (table)
SQL> ! just created in the repository:
SQL> !
SQL> CREATE TABLE FROM SYS$COMMON:[REPOSITORY]TABLE_TEST.PARTS
cont>        ALIAS AUTO;
SQL> !
SQL> ! Use the SHOW TABLE statement to display the information about the
SQL> ! PARTS table:
SQL> !
SQL> SHOW TABLE AUTO.PARTS;
Information for table AUTO.PARTS
```

```
CDD Pathname: SYS$COMMON:[REPOSITORY]TABLE_TEST.PARTS;1

Columns for table AUTO.PARTS:
Column Name   Data Type  Domain
-----------   ---------  ------
PART_NO                         SMALLINT  AUTO.PART_NO
PART_ID                         INTEGER   AUTO.PART_ID
PART_ID_USED_IN                 INTEGER   AUTO.PART_ID_USED_IN
PART_QUANT                      SMALLINT  AUTO.PART_QUANT

Table constraints for AUTO.PARTS:
AUTO.PARTS_PMK
 Primary Key constraint
 Table constraint for AUTO.PARTS
 Evaluated on each VERB
 Source: primary key PART_ID

AUTO.PARTS_UNQ
 Unique constraint
 Table constraint for AUTO.PARTS
 Evaluated on each VERB
 Source: unique PART_NO

AUTO.PART_CST
 Check constraint
 Table constraint for AUTO.PARTS
 Evaluated on each VERB
 Source: (ANY (P IN PARTS WITH (PART_ID IN PARTS EQ PART_ID_USED_IN IN P)))

AUTO.PART_FRK
 Foreign Key constraint
 Table constraint for AUTO.PARTS
 Evaluated on each VERB
 Source: foreign key PART_ID references OTHER_PARTS OTHER_PART_ID

Constraints referencing table AUTO.PARTS:
No constraints found
   .
   .
   .
SQL> --
SQL> COMMIT;
SQL> DISCONNECT DEFAULT;
SQL> EXIT;
```
♦

Example 8: Defining table-specific constraints with single-column primary and foreign keys

This example uses single-column keys to define table-specific constraints. The example maintains referential integrity among the four tables involved by using primary and foreign keys.

Three single-column primary key constraints preserve the integrity among the tables. The primary key constraints are the EMPLOYEE_ID column for the EMPLOYEES_TEST table, the JOB_CODE column for the JOBS_TEST table, and the DEPARTMENT_CODE column for the DEPARTMENTS_TEST table. The JOB_HISTORY_TEST table contains three foreign key constraints that refer to these primary keys.

Because the dialect is set to SQL92, constraints are NOT DEFERRABLE.

```
SQL> SET DIALECT 'SQL92';
SQL> --
SQL> CREATE TABLE EMPLOYEES_TEST
cont>   (EMPLOYEE_ID       ID_DOM
cont>        CONSTRAINT E_TEST_EMP_ID_PRIMARY
cont>        PRIMARY KEY,
cont>    LAST_NAME         LAST_NAME_DOM,
cont>    FIRST_NAME        FIRST_NAME_DOM,
cont>    MIDDLE_INITIAL    MIDDLE_INITIAL_DOM,
cont>    ADDRESS_DATA_1    ADDRESS_DATA_1_DOM,
cont>    ADDRESS_DATA_2    ADDRESS_DATA_2_DOM,
cont>    CITY              CITY_DOM,
cont>    STATE             STATE_DOM,
cont>    POSTAL_CODE       POSTAL_CODE_DOM,
cont>    SEX               SEX_DOM,
cont>    BIRTHDAY          DATE_DOM,
cont>    STATUS_CODE       STATUS_CODE_DOM);
SQL> --
SQL> CREATE TABLE JOBS_TEST
cont>   (JOB_CODE             JOB_CODE_DOM,
cont>        CONSTRAINT J_TEST_CODE_PRIMARY
cont>        PRIMARY KEY (JOB_CODE),
cont>    WAGE_CLASS           WAGE_CLASS_DOM,
cont>    JOB_TITLE            JOB_TITLE_DOM,
cont>    MINIMUM_SALARY       SALARY_DOM,
cont>    MAXIMUM_SALARY       SALARY_DOM);
SQL> --
SQL> CREATE TABLE DEPARTMENTS_TEST
cont>   (DEPARTMENT_CODE      DEPARTMENT_CODE_DOM,
cont>        CONSTRAINT D_DEPT_CODE_PRIMARY
cont>        PRIMARY KEY (DEPARTMENT_CODE),
cont>    DEPARTMENT_NAME      DEPARTMENT_NAME_DOM,
cont>    MANAGER_ID           ID_DOM,
cont>    BUDGET_PROJECTED     BUDGET_DOM,
cont>    BUDGET_ACTUAL        BUDGET_DOM);
SQL> --
SQL> CREATE TABLE JOB_HISTORY_TEST
cont>    (EMPLOYEE_ID         ID_DOM
cont>         CONSTRAINT JH_TEST_EMP_ID_FOREIGN
cont>         REFERENCES EMPLOYEES_TEST (EMPLOYEE_ID),
cont>     JOB_CODE            JOB_CODE_DOM
cont>         CONSTRAINT JH_J_CODE_FOREIGN
cont>         REFERENCES JOBS_TEST (JOB_CODE),
cont>    JOB_START            DATE_DOM,
cont>    JOB_END              DATE_DOM,
cont>    DEPARTMENT_CODE      DEPARTMENT_CODE_DOM
cont>         CONSTRAINT JH_D_CODE_FOREIGN
cont>         REFERENCES DEPARTMENTS_TEST (DEPARTMENT_CODE),
cont>    SUPERVISOR_ID        ID_DOM);
SQL>
```

Example 9: Defining table-specific constraints with multicolumn primary and foreign keys

The following example uses multicolumn keys to define table-specific constraints using a segment of the personnel database. This example uses some definitions not supplied with the sample database.

In this example, the two columns LOC and DEPT constitute a key, and they are defined as a PRIMARY KEY constraint for the WORK_STATION table. The two columns LOCATION and DEPARTMENT in the WORKER table are a foreign key that references the primary key in the WORK_STATION table.

Because the dialect is set to SQL92, constraints are NOT DEFERRABLE, and you do not receive a deprecated feature message when you define a constraint.

```
SQL> SET DIALECT 'SQL92';
SQL> --
SQL> CREATE DOMAIN LOC_DOM CHAR (10);
SQL> CREATE DOMAIN DEPT_DOM CHAR (10);
SQL> CREATE DOMAIN MGR_DOM CHAR (20);
SQL> CREATE DOMAIN NAME_DOM CHAR (20);
SQL> --
SQL> CREATE TABLE WORK_STATION
cont>   (LOC       LOC_DOM,
cont>    DEPT      DEPT_DOM,
cont>         CONSTRAINT WS_LOC_DEPT_PRIMARY
cont>         PRIMARY KEY (LOC, DEPT),
cont>    MGR       MGR_DOM);
SQL> --
SQL> CREATE TABLE WORKER
cont>   (NAME           NAME_DOM
cont>         CONSTRAINT WORKER_PRIMARY_NAME
cont>         PRIMARY KEY,
cont>    LOCATION       LOC_DOM,
cont>    DEPARTMENT     DEPT_DOM,
cont>         CONSTRAINT WORKER_FOREIGN_LOCATION_DEPT
cont>         FOREIGN KEY (LOCATION, DEPARTMENT)
cont>         REFERENCES WORK_STATION (LOC, DEPT));
SQL>
```

Example 10: Defining a table that contains a list

The following example defines a column of the data type LIST OF BYTE VARYING for storing employee resumes. This example defines the column EMPLOYEE_ID in the table EMPLOYEES as a foreign key constraint because resumes are kept only for actual employees for use in human resource management applications. Applications could use this table to identify employees with special backgrounds and skills for possible job assignments or promotions.

```
SQL> CREATE DOMAIN RESUME_DOM LIST OF BYTE VARYING;
SQL> CREATE TABLE RESUMES
cont>  (EMPLOYEE_ID      ID_DOM
cont>   REFERENCES EMPLOYEES (EMPLOYEE_ID),
cont>   RESUME            RESUME_DOM);
SQL> SHOW TABLE RESUMES;
Information for table RESUMES

Columns for table RESUMES:

Columns for table RESUMES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          ID_DOM
 Foreign Key constraint RESUMES_FOREIGN1
 Unique constraint RESUMES_UNIQUE_EMPLOYEE_ID
RESUME                         VARBYTE LIST    RESUME_DOM
                                        Segment Length: 1
```

```
Table constraints for RESUMES:
RESUMES_FOREIGN1
 Foreign Key constraint
 Column constraint for RESUMES.EMPLOYEE_ID
 Evaluated on COMMIT
 Source:
        RESUMES.EMPLOYEE_ID REFERENCES EMPLOYEES (EMPLOYEE_ID)

RESUMES_UNIQUE_EMPLOYEE_ID
 Unique constraint
 Column constraint for RESUMES.EMPLOYEE_ID
 Evaluated on COMMIT
 Source:
        RESUMES.EMPLOYEE_ID UNIQUE

Constraints referencing table RESUMES:
No constraints found

Indexes on table RESUMES:
No indexes found

Storage Map for table RESUMES:
     RESUMES_MAP

Triggers on table RESUMES:
No triggers found

SQL>
```

Example 11: Defining a table with a computed column that uses a select expression

You can use a select expression in a COMPUTED BY clause. The following example shows how to use the COMPUTED BY clause to count the number of current employees of a particular department.

```
SQL> CREATE TABLE DEPTS1
cont>       (DEPARTMENT_CODE DEPARTMENT_CODE_DOM,
cont>         DEPT_COUNT   COMPUTED BY
cont>         (SELECT COUNT (*) FROM  JOB_HISTORY JH
cont>            WHERE JOB_END IS NULL
cont>            AND
cont> --
cont> -- Use correlation names to qualify the DEPARTMENT_CODE columns.
cont>             DEPTS1.DEPARTMENT_CODE = JH.DEPARTMENT_CODE),
cont>         DEPARTMENT_NAME DEPARTMENT_NAME_DOM)
cont> ;
SQL> SELECT * FROM DEPTS1 WHERE DEPARTMENT_CODE = 'ADMN';
 DEPARTMENT_CODE    DEPT_COUNT   DEPARTMENT_NAME
 ADMN                        7   Corporate Administration
1 row selected
```

Example 12: Creating a table using the database default character set, national character set, and other character sets to define the columns

Assume the database was created defining the database default character set as DEC_KANJI and the national character set as KANJI.

```
SQL> CREATE TABLE COLOURS
cont>       (ENGLISH    MCS_DOM,
cont>        FRENCH     MCS_DOM,
cont>        JAPANESE   KANJI_DOM,
cont>        ROMAJI     DEC_KANJI_DOM,
cont>        KATAKANA   KATAKANA_DOM,
cont>        HINDI      HINDI_DOM,
cont>        GREEK      GREEK_DOM,
cont>        ARABIC     ARABIC_DOM,
cont>        RUSSIAN    RUSSIAN_DOM);
SQL> SHOW TABLE (COLUMNS) COLOURS;
Information for table COLOURS

Columns for table COLOURS:
Column Name                     Data Type        Domain
-----------                     ---------        ------
ENGLISH                         CHAR(8)          MCS_DOM
         DEC_MCS 8 Characters,  8 Octets
FRENCH                          CHAR(8)          MCS_DOM
         DEC_MCS 8 Characters,  8 Octets
JAPANESE                        CHAR(8)          KANJI_DOM
         KANJI 4 Characters,  8 Octets
ROMAJI                          CHAR(16)         DEC_KANJI_DOM
KATAKANA                        CHAR(8)          KATAKANA_DOM
         KATAKANA 8 Characters,  8 Octets
HINDI                           CHAR(8)          HINDI_DOM
         DEVANAGARI 8 Characters,  8 Octets
GREEK                           CHAR(8)          GREEK_DOM
         ISOLATINGREEK 8 Characters,  8 Octets
ARABIC                          CHAR(8)          ARABIC_DOM
         ISOLATINARABIC 8 Characters,  8 Octets
RUSSIAN                         CHAR(8)          RUSSIAN_DOM
         ISOLATINCYRILLIC 8 Characters,  8 Octets
```

Example 13: Creating and using a global temporary table

Assume that you have a base table called PAYROLL that is populated with data and that you want to extract the current week's information to generate paychecks for the company. The following example shows how to create a global temporary table called PAYCHECKS_GLOB and populate it with data from the PAYROLL and EMPLOYEES base tables. Your application can now operate on the data in PAYCHECKS_GLOB to calculate deductions and net pay for each employee. This eliminates continuous queries to the base tables and reduces concurrency conflicts.

```
SQL> CREATE GLOBAL TEMPORARY TABLE PAYCHECKS_GLOB
cont>       (EMPLOYEE_ID ID_DOM,
cont>        LAST_NAME CHAR(14),
cont>        HOURS_WORKED INTEGER,
cont>        HOURLY_SAL   INTEGER(2),
cont>        WEEKLY_PAY   INTEGER(2))
cont>        ON COMMIT PRESERVE ROWS;
SQL> --
SQL> -- Insert data into the temporary tables from other existing tables.
SQL> INSERT INTO PAYCHECKS_GLOB
cont>         (EMPLOYEE_ID, LAST_NAME, HOURS_WORKED, HOURLY_SAL, WEEKLY_PAY)
cont>          SELECT P.EMPLOYEE_ID, E.LAST_NAME, P.HOURS_WORKED, P.HOURLY_SAL,
cont>                 P.HOURS_WORKED * P.HOURLY_SAL
cont>                 FROM EMPLOYEES E, PAYROLL P
cont>                 WHERE E.EMPLOYEE_ID = P.EMPLOYEE_ID
cont>                   AND P.WEEK_DATE = DATE '1995-08-01';
100 rows inserted
SQL> --
SQL> -- Display the data.
SQL> SELECT * FROM PAYCHECKS_GLOB LIMIT TO 2 ROWS;
 EMPLOYEE_ID   LAST_NAME       HOURS_WORKED      HOURLY_SAL       WEEKLY_PAY
 00165         Smith                     40           30.50          1220.00
 00166         Dietrich                  40           36.00          1440.00
2 rows selected
SQL> -- Commit the data.
SQL> COMMIT;
SQL> --
SQL> -- Because the global temporary table was created with PRESERVE ROWS,
SQL> -- the data is preserved after you commit the transaction.
SQL> SELECT * FROM PAYCHECKS_GLOB LIMIT TO 2 ROWS;
 EMPLOYEE_ID   LAST_NAME       HOURS_WORKED      HOURLY_SAL       WEEKLY_PAY
 00165         Smith                     40           30.50          1220.00
 00166         Dietrich                  40           36.00          1440.00
2 rows selected
```

## CREATE TRIGGER Statement

Creates triggers for a specified table. A **trigger** defines the actions to occur before or after the table is updated (by a write operation such as an INSERT, DELETE, or UPDATE statement). The trigger is associated with a single table, takes effect at a specific time for a particular type of update, and causes one or more triggered actions to be performed. If the trigger specifies multiple actions, each action is performed in the order in which it appears within the trigger definition.

With triggers, you can define useful actions such as:

- Cascading deletes

  Deleting a row from one table causes additional rows to be deleted from other tables that are related to the first table by key values.

- Cascading updates

  Updating a row in one table causes additional rows to be updated in other tables that are related to the first table by key values. These updates are commonly limited to the key fields themselves.

- Summation updates

  Updating a row from one table causes a value in a row of another table to be updated by being increased or decreased.

- Hidden deletes

  Causing rows to be deleted from a table by moving them to a parallel table that is not otherwise used by the database.

**Note**

Combinations of table-specific constraints and appropriately defined triggers, by themselves, are not sufficient to guarantee that database integrity is preserved when the database is updated. If integrity is to be preserved, table-specific constraints and triggers must be used in conjunction with a common set of update procedures that ensure completely reproducible and consistent retrieval and update strategies.

The CREATE TRIGGER statement adds the trigger definition to the physical database.

If you did not attach to the database by a path name, the trigger definition is not stored in the repository. This causes an inconsistency between the definitions in the database and the repository. Therefore, you must define the triggers again whenever you restore the database metadata from the repository using the INTEGRATE statement.

A **triggered action** consists of an optional predicate and some triggered statements. If specified, the predicate must evaluate to true for the triggered statements in the action to execute. Each triggered statement is executed in the order in which it appears within the triggered action clause.

The triggered statement can be:

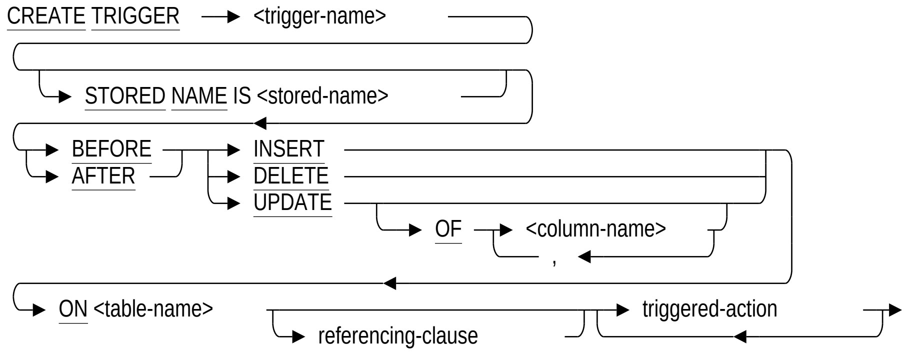
- A DELETE statement
- An UPDATE statement
- An INSERT statement
- An ERROR statement

## Environment

You can use the CREATE TRIGGER statement:

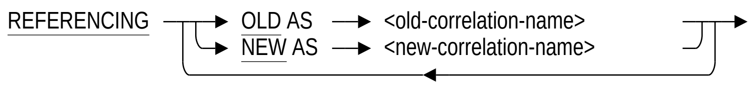
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

## Format

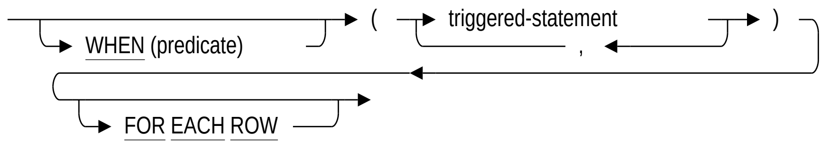


## CREATE TRIGGER Statement

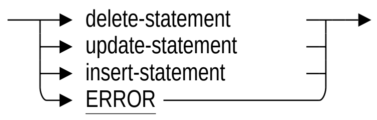
referencing-clause =



triggered-action =



triggered-statement =



## Arguments

**trigger-name**
The name of the trigger being defined. The name must be unique within the database.

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access a trigger created in a multischema database. The stored name allows you to access multischema definitions using interfaces, such as Oracle RMU, the Oracle Rdb management utility, that do not recognize multiple schemas in one database. You cannot specify a stored name for a trigger in a database that does not allow multiple schemas. For more information on stored names, see Section 2.2.4.

**column-name**
The name of a column within the specified table to be checked for deletion, modification, or insertion. Use this argument only with UPDATE triggers.

**table-name**
The name of the table for which this trigger is defined.

**referencing-clause**
Lets you specify whether you want to refer to the row values as they existed before an UPDATE operation occurred or the new row values after they are applied by the UPDATE operation. Do not use this clause with INSERT or DELETE operations.

You can specify each option (OLD AS old-correlation-name or NEW AS new-correlation-name) only once in the referencing clause.

**old-correlation-name**
A temporary name used to refer to the row values as they existed before an UPDATE operation occurred. If you do not specify the FOR EACH ROW clause, this correlation name cannot be referred to in the triggered statement.

**new-correlation-name**
A temporary name used to refer to the new row values to be applied by the UPDATE operation. If you do not specify the FOR EACH ROW clause, this correlation name cannot be referred to in the triggered statement.

**triggered-action**
Consists of an optional predicate, some triggered statements, and an optional frequency clause. If specified, the predicate must evaluate to true for the triggered statements in the triggered action clause to execute. Each triggered statement is executed in the order in which it appears within the triggered action clause.

**WHEN (predicate)**
Describes the optional condition that must be satisfied before the associated triggered statements are executed. This predicate cannot refer to any host language variable.

To avoid ambiguity between columns and external function callouts, use parentheses around the predicate in the WHEN clause. See the Usage Notes for further explanation.

**triggered-statement**
Updates the database, rolls back a transaction, or generates an error message.

**delete-statement**
Specifies the row of a table that you want to delete. If you specify CURRENT OF cursor-name with the WHERE clause of the DELETE statement, you receive an error message.

When you use the DELETE statement as a triggered statement, omit the semicolon that normally terminates a DELETE statement.

**update-statement**
Specifies the row of a table that you want to modify. If you specify CURRENT OF cursor-name with the WHERE clause of the UPDATE statement, you receive an error message.

When you use the UPDATE statement as a triggered statement, omit the semicolon that normally terminates an UPDATE statement.

**insert-statement**
Specifies the new row or rows you want to add to a table.

When you use the INSERT statement as a triggered statement, omit the semicolon that normally terminates an INSERT statement.

**ERROR**
Provides the following message:

```
RDMS-E-TRIG_ERROR, Trigger 'trigger_name' forced an error.
```

A triggered ERROR statement cancels the UPDATE statement that invoked the trigger.

**FOR EACH ROW**
Determines whether the triggered action is evaluated once per triggering statement, or for each row of the subject table that is affected by the triggering statement. If the FOR EACH ROW clause is not specified, the triggered action is evaluated only once, and row values are not available to the triggered action.

## Usage Notes

- Triggers can be used to help ensure referential integrity among tables. For every value of a foreign key in one table, a trigger ensures a matching value in the primary key field of another table.
- Triggers cannot reference stored functions or stored procedures.
- If you use a view name in any part of a CREATE TRIGGER statement, you receive the following error:

  ```
  RDB-E-NO_META_UPDATE, metadata update failed
  -RDMS-F-TRGVW, views cannot be used within a trigger
  ```

  Use the tables in the view rather than the actual view name.

- Creating a trigger requires SELECT and CREATE access to the subject table, and if any triggered statement specifies some form of update operation, also requires SELECT, DBCTRL, and the appropriate type of update (DELETE, UPDATE, INSERT) access to the tables specified by the triggered action statement.
- The trigger specification includes an action time, an update event (some type of write operation to the database), and an optional column list, which together determine when the trigger is to be evaluated. The action time can be specified as either before or after the update event (the INSERT, DELETE, or UPDATE statement). For triggers evaluated on UPDATE statements, you can specify an optional list of columns (from the subject table) to further stipulate that the trigger is to be evaluated only when one of the columns listed is also listed in the SET column list of the UPDATE statement. The trigger will be evaluated whether or not the values within the listed columns are actually changed during the execution of the UPDATE statement.
- Appropriate conditions may be placed in the WHEN predicate using both the NEW and OLD context values to prevent the execution of the trigger action if the actual column values did not change during the update.
- The frequency clause, FOR EACH ROW, determines whether an action is evaluated once per triggering statement, or for each row of the subject table that is affected by the triggering statement. If the FOR EACH ROW clause is not specified, the action is evaluated only once, and row values are not available to the triggered action.
- The table correlation name (current correlation name), old correlation name, and new correlation name are for various states of the subject table context of the triggered statement. The old correlation name is available (valid) only for AFTER UPDATE triggers and the new correlation name is available (valid) only for BEFORE UPDATE triggers.
- Only one trigger specifying one of the six combinations of action time and type of update statement can be defined for any table, with the exception that multiple BEFORE UPDATE or AFTER UPDATE triggers can be defined as long as they all have exclusive, unique column lists. The trigger being defined checks for conflicts with the specified trigger for either update time and type, or in one of the column names on the list of columns to be modified. A triggered statement cannot affect the table on which the trigger is defined such that the trigger would be recursively invoked.

- Table 6–5 lists the six possible types of update action. Only one trigger specifying one of the six combinations of action time and type of update statement can be defined for any table. For update type UPDATE, this uniqueness is further qualified by any specified column names. A triggered statement cannot affect the table on which the trigger is defined such that the trigger would be recursively invoked.

  The values from the row affected by the triggering statement are available to the triggered actions, as shown in Table 6–5.

**Table 6–5 Availability of Row Data for Triggered Actions**

| Action Time/Type of Update | Availability of Row Data |
|---|---|
| BEFORE INSERT | Row data is not available. |
| AFTER INSERT | Row data referred to by the table correlation name is available. |
| BEFORE DELETE | Row data referred to by the table correlation name is available. |
| AFTER DELETE | Row data is not available. |
| BEFORE UPDATE | Old values of row data referred to by the table correlation name are available. |
| | New values of row data referred to by the new correlation name are available. |
| AFTER UPDATE | New values of row data referred to by the table correlation name are available. |
| | Old values of row data referred to by the old correlation name are available. |

  If the FOR EACH ROW clause is not specified, the triggered action is evaluated only once, and row values are not available to the triggered action.

  For example, a BEFORE INSERT trigger action for the EMPLOYEES table cannot create a row in the JOB_HISTORY table for the ID in the EMPLOYEE_ID column to be stored because the information in the row to be stored is not yet available. However, an AFTER INSERT trigger action can use the EMPLOYEE_ID column of the row being stored to create a row in the JOB_HISTORY table.

A BEFORE DELETE trigger action for the EMPLOYEES table can delete rows in the JOB_HISTORY table using the EMPLOYEE_ID column of the row to be deleted. However, an AFTER DELETE trigger action cannot delete any JOB_HISTORY rows using that EMPLOYEE_ID column because the information from the deleted row is no longer available.

- Once a trigger is selected for evaluation, SQL evaluates each pertinent triggered action in succession. The execution of a triggered action statement may cause other triggers to be selected for invocation; however, if a trigger is selected recursively by a direct or indirect execution of one of its actions, an exception is produced. Once all triggered actions have been exhausted, another pertinent trigger may be selected for evaluation (BEFORE UPDATE and AFTER UPDATE triggers only).
- An existing trigger cannot be changed. If you want to modify an existing trigger, you must delete it, then create a new trigger.
- The number in the third element of the SQLERRD array, SQLERRD[2], and the number displayed at the end of a statement in interactive SQL do not include the rows inserted, updated, and deleted by triggers.
- You must execute the CREATE TRIGGER statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a read/write transaction implicitly.
- Attempts to create a trigger fail if that trigger or its affected tables are involved in a query at the same time. Users must detach from the database with a DISCONNECT statement before you can create the trigger. When Oracle Rdb first accesses an object such as the table, a lock is placed on that object and not released until the user exits the database. If you attempt to update this object, you get a LOCK CONFLICT ON CLIENT message due to the other user’s access to the object.

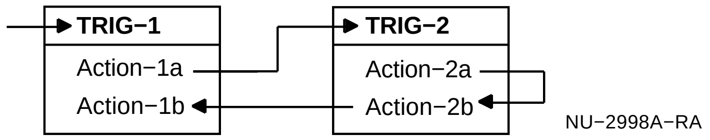
  Similarly, while you create a trigger, users cannot execute queries involving that trigger or its tables until you completed the transaction with a COMMIT or ROLLBACK statement for the CREATE statement. The user receives a LOCK CONFLICT ON CLIENT error message. While DDL operations are performed, normal data locking mechanisms are used against system tables. (System tables contain information about objects in the database.) Therefore, attempts to update an object lock out attempts to query that object. These locks are held until the execution of a COMMIT or ROLLBACK statement in the DDL operation.

The WAIT/NOWAIT clause of the SET TRANSACTION statement does not affect attempts to update metadata with simultaneous queries. Even if you specify SET TRANSACTION WAIT for the metadata update transaction, you get the following error message if a lock conflict exists:

```
%RDB-E-LOCK_CONFLICT, request failed due to locked resource; no-wait
parameter specified for transaction
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-LCKCNFLCT, lock conflict on client
SQL>
```

However, a user's query waits for a metadata update to complete with a ROLLBACK or COMMIT statement, even if the user specified NOWAIT in the SET TRANSACTION statement.

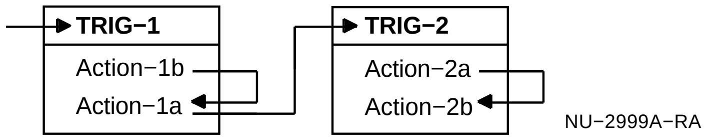
- You cannot execute the CREATE TRIGGER statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on the RDB$SYSTEM storage area.
- Other users are allowed to be attached to the database when you issue the CREATE TRIGGER statement.
- If a trigger references a table not specified in the RESERVING clause of the SET TRANSACTION statement, that table is reserved as SHARED WRITE. If the table referenced by a trigger is already reserved in an incompatible mode, the statement that activates it fails.
- If you invoke a trigger performing more than one action and one of those actions invokes another trigger, the actions performed in the second trigger must complete before the subsequent actions of the first trigger are executed. For example:



When TRIG-1 is invoked, Action-1a is executed which invokes TRIG-2. All actions of TRIG-2 must complete before any subsequent actions of TRIG-1 can execute. The actions of TRIG-1 and TRIG-2 occur in the following order:

Action-1a
Action-2a
Action-2b
Action-1b

The actions of TRIG-2 are not affected by the results of Action-1b because Action-1b does not execute until TRIG-2 is complete. Should you need the result of Action-1b to affect the results of TRIG-2, reverse the actions in TRIG-1. For example:



The actions of TRIG-1 and TRIG-2 now occur in the following order:

Action-1b
Action-1a
Action-2a
Action-2b

- The inclusion of an external function callout in a value expression causes ambiguity with conditional trigger definitions in V6.0 and earlier.

  For example, the following syntax is ambiguous:

```
    .
    .
    .
WHEN '00190' <> EMPLOYEE_ID (ERROR)
    .
    .
    .
```

  In the preceding example, it is difficult to determine if the predicate refers to the column EMPLOYEE_ID followed by an action or error, or if the predicate refers to a function callout to the function EMPLOYEE_ID with an argument of ERROR. To support function callouts within trigger definitions, SQL assumes this is a function callout. For existing trigger definitions, this causes a parsing error.

  Use parentheses around the predicate in the WHEN clause to avoid this ambiguity.

## Examples

Example 1: Defining a cascading delete trigger

The following SQL procedure shows a trigger from the sample personnel database that deletes rows in several tables before deleting a row in the EMPLOYEES table. Each associated employee record (from the tables that have foreign keys referring to the primary key in the employee record) is deleted. The employee identification number being deleted (00164) belongs to an employee who is also a manager; therefore, the MANAGER_ID column in the DEPARTMENTS table is set to null, as specified by the trigger.

```
SQL> SET TRANSACTION READ WRITE;
SQL> --
SQL> -- Display the EMPLOYEE_ID_CASCADE_DELETE trigger
SQL> -- in the sample database:
SQL> --
SQL> SHOW TRIGGER EMPLOYEE_ID_CASCADE_DELETE
     EMPLOYEE_ID_CASCADE_DELETE
 Source:
   EMPLOYEE_ID_CASCADE_DELETE
                       BEFORE DELETE ON EMPLOYEES
                       (DELETE FROM DEGREES D WHERE D.EMPLOYEE_ID =
                        EMPLOYEES.EMPLOYEE_ID)
                          FOR EACH ROW
                       (DELETE FROM JOB_HISTORY JH WHERE JH.EMPLOYEE_ID =
                        EMPLOYEES.EMPLOYEE_ID)
                          FOR EACH ROW
                       (DELETE FROM SALARY_HISTORY SH WHERE SH.EMPLOYEE_ID =
                        EMPLOYEES.EMPLOYEE_ID)
                          FOR EACH ROW
                     -- Also, if an employee is terminated and that employee
                     -- is the manager of a department, set the MANAGER_ID
                     -- column to null for that department.
                       (UPDATE DEPARTMENTS D  SET D.MANAGER_ID = NULL
                        WHERE D.MANAGER_ID = EMPLOYEES.EMPLOYEE_ID)
                          FOR EACH ROW
SQL> --
SQL> -- The EMPLOYEES table has a value of "00164"
SQL> -- in the EMPLOYEE_ID column:
SQL> --
SQL> SELECT * FROM EMPLOYEES E WHERE E.EMPLOYEE_ID = "00164";
 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

1 row selected
SQL> --
SQL> --
SQL> -- The DEGREES table has two values of "00164"
SQL> -- in the EMPLOYEE_ID column:
SQL> --
SQL> SELECT * FROM DEGREES D WHERE D.EMPLOYEE_ID = "00164";
 EMPLOYEE_ID   COLLEGE_CODE   YEAR_GIVEN   DEGREE   DEGREE_FIELD
 00164         PRDU                 1982   PhD      Statistics
 00164         PRDU                 1973   MA       Applied Math
2 rows selected
SQL> --
SQL> --
SQL> -- The JOB_HISTORY table has the value of "00164" in
SQL> -- several rows in the EMPLOYEE_ID column:
SQL> --
SQL> SELECT * FROM JOB_HISTORY JH WHERE JH.EMPLOYEE_ID = "00164";
 EMPLOYEE_ID   JOB_CODE   JOB_START     JOB_END       DEPARTMENT_CODE
   SUPERVISOR_ID
 00164         DMGR       21-Sep-1981   NULL          MBMN
   00228

 00164         SPGM        5-Jul-1980   20-Sep-1981   MCBM
   00164

2 rows selected
SQL> --
SQL> --
SQL> -- The SALARY_HISTORY table has a value of "00164"
SQL> -- in several rows in the EMPLOYEE_ID column:
SQL> --
SQL> SELECT * FROM SALARY_HISTORY SH WHERE SH.EMPLOYEE_ID = "00164";
 EMPLOYEE_ID   SALARY_AMOUNT   SALARY_START   SALARY_END
 00164            $50,000.00   21-Sep-1981    14-Jan-1983
 00164            $26,291.00    2-Mar-1981    21-Sep-1981
 00164            $51,712.00   14-Jan-1983    NULL
 00164            $26,291.00    5-Jul-1980     2-Mar-1981
4 rows selected
SQL> --
SQL> --
SQL> -- The DEPARTMENTS table has a value of "00164"
SQL> -- in the MANAGER_ID column:
SQL> --
SQL> SELECT * FROM DEPARTMENTS D WHERE D.MANAGER_ID = "00164";
 DEPARTMENT_CODE   DEPARTMENT_NAME                 MANAGER_ID
   BUDGET_PROJECTED   BUDGET_ACTUAL
 MBMN              Board Manufacturing North       00164
               NULL            NULL

1 row selected
SQL> --
SQL> --
SQL> -- Test the trigger by deleting the row with a value of "00164"
SQL> -- in the EMPLOYEE_ID column from the EMPLOYEES table:
SQL> --
SQL> DELETE FROM EMPLOYEES E WHERE E.EMPLOYEE_ID = "00164";
1 row deleted
SQL> --
SQL> -- The row with a value of "00164" in the EMPLOYEE_ID column
SQL> -- was deleted from the EMPLOYEES table:
SQL> --
SQL> SELECT * FROM EMPLOYEES E WHERE E.EMPLOYEE_ID = "00164";
0 rows selected
SQL> --
SQL> -- The rows with a value of "00164" in the EMPLOYEE_ID column
SQL> -- were deleted from the DEGREES table:
SQL> --
SQL> SELECT * FROM DEGREES D WHERE D.EMPLOYEE_ID = "00164";
0 rows selected
SQL> --
SQL> -- The rows with a value of "00164" in the EMPLOYEE_ID
SQL> -- column were deleted from the JOB_HISTORY table:
SQL> --
SQL> SELECT * FROM JOB_HISTORY JH WHERE JH.EMPLOYEE_ID = "00164";
0 rows selected
SQL> --
SQL> -- The rows with a value of "00164" in the EMPLOYEE_ID
SQL> -- column were deleted from the SALARY_HISTORY table:
SQL> --
SQL> SELECT * FROM SALARY_HISTORY SH WHERE SH.EMPLOYEE_ID = "00164";
0 rows selected
SQL> --
SQL> -- The value of "00164" in the MANAGER_ID column was set to null
SQL> -- in the DEPARTMENTS table:
SQL> --
SQL> SELECT * FROM DEPARTMENTS D WHERE D.DEPARTMENT_CODE = "MBMN";
 DEPARTMENT_CODE   DEPARTMENT_NAME                  MANAGER_ID
   BUDGET_PROJECTED   BUDGET_ACTUAL
 MBMN             Board Manufacturing North        NULL
               NULL            NULL

1 row selected
SQL> --
SQL> ROLLBACK;
SQL> EXIT
```

Example 2: Defining a trigger that performs an update

Before the STATUS_CODE column in WORK_STATUS table is updated, the STATUS_CODE_CASCADE_UPDATE trigger in the following SQL procedure updates the associated rows in the EMPLOYEES table. The REFERENCING clause specifies OLD_WORK_STATUS as the correlation name for the values in the WORK_STATUS table before the UPDATE statement executes, and NEW_

WORK_STATUS as the correlation name for the values in the WORK_STATUS table after the UPDATE statement executes.

```
SQL> -- Display the STATUS_CODE_CASCADE_UPDATE trigger in
SQL> -- the sample database:
SQL> --
SQL> SHOW TRIGGER STATUS_CODE_CASCADE_UPDATE
     STATUS_CODE_CASCADE_UPDATE
 Source:
   STATUS_CODE_CASCADE_UPDATE
                       BEFORE UPDATE OF STATUS_CODE ON WORK_STATUS
                          REFERENCING OLD AS OLD_WORK_STATUS
                                      NEW AS NEW_WORK_STATUS
                       (UPDATE EMPLOYEES E
                        SET E.STATUS_CODE = NEW_WORK_STATUS.STATUS_CODE
                        WHERE E.STATUS_CODE = OLD_WORK_STATUS.STATUS_CODE)
                          FOR EACH ROW
SQL> --
SQL> -- Change the STATUS_CODE column with a value of 2 to a value of 3:
SQL> --
SQL> UPDATE WORK_STATUS WS SET STATUS_CODE="3" WHERE STATUS_CODE="2";
1 row updated
SQL> --
SQL> -- The trigger changes any STATUS_CODE column in the EMPLOYEES table
SQL> -- with a value of 2 to a value of 3. Therefore, no rows are
SQL> -- selected for the first query that follows, but several are selected
SQL> -- for the second query:
SQL> --
SQL> SELECT * FROM EMPLOYEES E WHERE E.STATUS_CODE = "2";
0 rows selected
SQL> --
SQL> SELECT * FROM EMPLOYEES E WHERE E.STATUS_CODE = "3";
 EMPLOYEE_ID   LAST_NAME       FIRST_NAME   MIDDLE_INITIAL
   ADDRESS_DATA_1             ADDRESS_DATA_2         CITY
      STATE   POSTAL_CODE   SEX    BIRTHDAY      STATUS_CODE
 00165         Smith           Terry        D
   120 Tenby Dr.                                     Chocorua
      NH      03817         M      15-May-1954   3

 00178         Goldstone       Neal         NULL
   194 Lyons Av,                                     Colebrook
      NH      03576         M      25-Apr-1952   3

   .
   .
   .
 00358         Lapointe        Jo Ann       C
   70 Tenby Dr.                                      Chocorua
      NH      03817         F      24-Feb-1931   3
```

```
12 rows selected
SQL> --
SQL> ROLLBACK;
```

Example 3: Defining a trigger that updates a sales summary

The following example defines a trigger that updates a monthly sales total after each daily sale is made.

```
SQL> --
SQL> -- Create the table to keep track of monthly sales:
SQL> CREATE TABLE MONTHLY_SALES
cont> ( SALES_AMOUNT INTEGER);
SQL> --
SQL> -- Create the table to keep track of sales made today:
SQL> CREATE TABLE DAILY_SALES
cont> ( SALES_AMOUNT INTEGER);
SQL> --
SQL> -- Assume that $250.00 of sales have been made during the current month:
SQL> INSERT INTO MONTHLY_SALES
cont> (SALES_AMOUNT) VALUES (250);
1 row inserted
SQL> --
SQL> -- After adding a new value to the SALES_AMOUNT column in
SQL> -- DAILY_SALES table, SQL updates the SALES column in
SQL> -- the MONTHLY_SALES table with the amount of the new sale:
SQL> CREATE TRIGGER UPDATE_SALES_TOTAL_ON_NEW_SALE
cont> AFTER INSERT ON DAILY_SALES
cont>   (UPDATE MONTHLY_SALES M
cont>       SET M.SALES_AMOUNT = M.SALES_AMOUNT + DAILY_SALES.SALES_AMOUNT)
cont> FOR EACH ROW;
SQL> --
SQL> -- The following statement records a new $5.00 sale for today:
SQL> INSERT INTO DAILY_SALES
cont> (SALES_AMOUNT) VALUES (5);
1 row inserted
SQL> --
SQL> -- The value for the SALES_AMOUNT column of the DAILY_SALES table
SQL> -- is $5.00 and the value of the SALES_AMOUNT column of the
SQL> -- MONTHLY_SALES table is $255.00:
SQL> SELECT * FROM DAILY_SALES;
 SALES_AMOUNT
            5
1 row selected
SQL> --
SQL> SELECT * FROM MONTHLY_SALES;
 SALES_AMOUNT
          255
1 row selected
SQL> --
SQL> -- When a new $9.00 sale is made, the values in the two rows of the
SQL> -- SALES_AMOUNT column of the DAILY_SALES table are $5.00 and $9.00
SQL> -- and the value of the SALES_AMOUNT column of the MONTHLY_SALES
SQL> -- table is $264.00:
SQL> INSERT INTO DAILY_SALES
cont> (SALES_AMOUNT) VALUES (9);
1 row inserted
SQL> --
SQL> SELECT * FROM DAILY_SALES;
 SALES_AMOUNT
            5
            9
2 rows selected
SQL> --
SQL> SELECT * FROM MONTHLY_SALES;
 SALES_AMOUNT
          264
1 row selected
SQL> --
SQL> ROLLBACK;
SQL> --
```

Example 4: Defining a trigger that sets column values to null

Before the STATUS_CODE column in the WORK_STATUS table is deleted, this trigger causes the associated WORK_STATUS columns in the EMPLOYEES table to be set to null.

```
SQL> CREATE TRIGGER STATUS_CODE_ON_DELETE_SET_NULL
cont>  BEFORE DELETE ON WORK_STATUS
cont>   (UPDATE EMPLOYEES E  SET E.STATUS_CODE = NULL
cont>    WHERE E.STATUS_CODE = WORK_STATUS.STATUS_CODE)
cont>  FOR EACH ROW;
SQL> --
SQL> -- Delete any row in the WORK_STATUS table where the STATUS_CODE
SQL> -- column has a value of 1:
SQL> DELETE FROM WORK_STATUS WS WHERE WS.STATUS_CODE = "1";
1 row deleted
SQL> --
SQL> -- This trigger sets the STATUS_CODE column value to null in many
SQL> -- rows in the EMPLOYEES table:
SQL> SELECT * FROM EMPLOYEES E WHERE E.STATUS_CODE IS NULL;
 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
   61 Broad st.                NULL                   Alton
      NH      03809         M      13-Apr-1941   NULL
 00374         Andriola         Leslie       Q
   111 Boston Post Rd.         NULL                   Salisbury
      NH      03268         M      19-Mar-1955   NULL
   .
   .
   .
 00200         Ziemke           Al           F
    121 Putnam Hill Rd.         NULL                   Winnisquam
          NH      03289         M      27-Oct-1928   NULL
88 rows selected
SQL> ROLLBACK;
```

Example 5: Defining a trigger that prevents deletion of a row that exists in two tables

Suppose that a user wants to delete only those rows in the JOB_HISTORY table that do not also exist in the JOBS table. This is difficult to do with constraints because a row can exist in one table with a key number that does not exist in the other table. The following statement creates a trigger that causes an error when the user tries to delete a row that exists in table JOB_ HISTORY.

```
SQL> CREATE TRIGGER DELETE_GUARD
cont> BEFORE DELETE ON JOB_HISTORY
cont> WHEN EXISTS (SELECT JOBS.JOB_CODE FROM JOBS
cont> WHERE JOBS.JOB_CODE=JOB_HISTORY.JOB_CODE)
cont> (ERROR) FOR EACH ROW;
SQL> --
SQL> -- Now attempt a deletion that violates the trigger.
SQL> --
SQL> DELETE FROM JOB_HISTORY WHERE JOB_CODE = 'DMGR';
%RDB-E-TRIG_INV_UPD, invalid update; encountered error condition
defined for trigger
-RDMS-E-TRIG_ERROR, trigger DELETE_GUARD forced an error
-RDB-F-ON_DB, on database DISK1:[DEPT3.SQL]MF_PERSONNEL.RDB;1
```

## CREATE VIEW Statement

Creates a view definition. A **view** is a logical structure that refers to rows stored in other tables. Data in a view is not physically stored in the database. You can include in a view definition combinations of rows and columns from other tables and view definitions in the schema. You define a view by specifying a select expression, that:

- Names the criteria for selecting the tables, rows, and columns for the view
- Specifies a set of columns from those tables

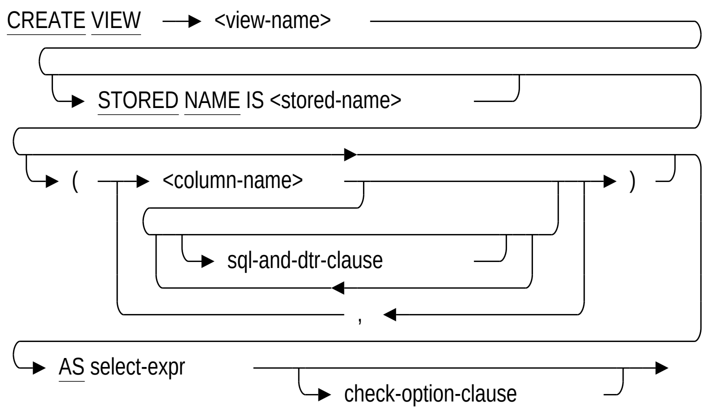
When the CREATE VIEW statement executes, SQL adds the view definition to the physical database. If you declared the schema with the PATHNAME argument, the definition is also stored in the repository.

### Environment

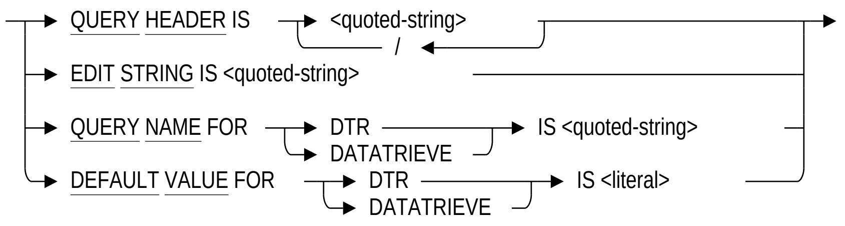
You can use the CREATE VIEW statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

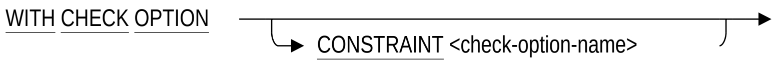
### Format



sql-and-dtr-clause =



check-option-clause =



## Arguments

**view-name**
Name of the view definition you want to create. When choosing a name, follow these rules:

- Use a name that is unique among all view and table names in the schema.
- Use any valid SQL name (see Section 2.2 for more information).

**STORED NAME IS stored-name**
Specifies a name that Oracle Rdb uses to access a view created in a multischema database. The stored name allows you to access multischema definitions using interfaces, such as Oracle RMU, the Oracle Rdb management utility, that do not recognize multiple schemas in one database. You cannot specify a stored name for a view in a database that does not allow multiple schemas. For more details about stored names, see Section 2.2.4.

**column-name**
A list of names for the columns of the view. If you omit column names, SQL assigns the names from the columns in the source tables in the select expression.

However, you must specify names for all the columns of the view in the following cases:

- The select expression generates columns with duplicate names.

- The select expression uses statistical functions or arithmetic expressions to create new columns that are not in the source tables.

**sql-and-dtr-clause**
Optional SQL and DATATRIEVE formatting clauses. See Section 2.5 for more information on formatting clauses.

You cannot use the clauses beginning with *NO* with the CREATE VIEW statement. They are valid only with the ALTER TABLE and ALTER DOMAIN statements.

**select-expr**
A select expression that defines which columns and rows of the specified tables SQL includes in the view. The select expression for a nonmultischema database can name only tables in the same schema as the view. A select expression for a multischema database can name a table in any schema in the database; the schema need not be in the same catalog as the view being created. See Section 2.8.1 for more information on select expressions.

**check-option-clause**
A constraint that places restrictions on update operations made to a view. The check option clause ensures that any rows that are inserted or updated in a view conform to the definition of the view. Do not specify the WITH CHECK OPTION clause with views that are read-only. (The Usage Notes describe which views SQL considers read-only.)

**CONSTRAINT check-option-name**
Specify a name for the WITH CHECK OPTION constraint. If you omit the name, SQL creates a name. However, Oracle Rdb recommends that you always name constraints. The constraint names generated by SQL may be obscure and, in programs, may change between compile time and run time. If you supply a name for the WITH CHECK OPTION constraint, the name must be unique in the schema.

The name for the WITH CHECK OPTION constraint is used by the INTEG_ FAIL error message when an INSERT or UPDATE statement violates the constraint.

## Usage Notes

You must execute the CREATE VIEW statement in a read/write transaction. If you issue this statement when there is no active transaction, SQL starts a transaction with characteristics specified in the most recent DECLARE TRANSACTION statement.

You cannot execute the CREATE VIEW statement when the RDB$SYSTEM storage area is set to read-only. You must first set RDB$SYSTEM to read/write. See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information on the RDB$SYSTEM storage area.

Any statement that inserts, updates, or deletes rows of a view changes the rows of the base tables on which the view is based. In general, avoid using views in INSERT, UPDATE, or DELETE statements.

Note the following when using INSERT, UPDATE, and DELETE statements that refer to views:

- Do not refer to read-only views in INSERT, UPDATE, or DELETE statements. SQL considers as read-only views those with select expressions that:
  - Use the DISTINCT argument to eliminate duplicate rows from the result table
  - Name more than one table or view in the FROM clause
  - Include a function in the select list
  - Contain a UNION, GROUP BY, or HAVING clause
- In INSERT and UPDATE statements, you cannot refer to columns in views that are the result of an arithmetic expression or a function. For instance, you cannot use an INSERT statement that refers to ARITH_COLUMN in the following view definition:

```
SQL> CREATE VIEW TEMP (ARITH_COLUMN, EMPLOYEE_ID)
cont>   AS SELECT (SALARY_AMOUNT * 3), EMPLOYEE_ID
cont>   FROM SALARY_HISTORY;
SQL>
SQL> INSERT INTO TEMP (ARITH_COLUMN) VALUES (111);
%RDB-E-READ_ONLY_FIELD, attempt to update read-only field ARITH_COLUMN
SQL> ROLLBACK;
```

- To allow correct SQLSTATE handling for the ANSI/ISO SQL standard, the exception raised by a WITH CHECK OPTION violation changes when the dialect is set to SQL92 at database attach time. For example:

  ```
  SQL> SET DIALECT 'SQL92';
  SQL> ATTACH 'FILENAME personnel_test';
  SQL> INSERT INTO MANAGERS VALUES (1, 'Fred', 10);
  %RDB-E-CHECK_FAIL, violation of view check option "MANAGERS_CHECKOPT1"
  caused operation to fail
  ```

  This change allows SQL to return a special SQLSTATE value of 44000 and allows applications to distinguish between constraint and view-check option violations. Adjust any error handlers that examine the RDB$MESSAGE_VECTOR so that they correctly handle RDB$_CHECK_FAIL (it is similar to the error RDB$_INTEG_FAIL). For more information about SQLSTATE values, see Appendix C.

- Use the WITH CHECK OPTION clause to make sure that rows you insert or update in a view conform to its definition.

  For example, the following view definition allows only salaries over $60,000. Because you use the WITH CHECK OPTION clause, you cannot insert a row that contains a salary of less than $60,000.

  ```
  SQL> CREATE VIEW TEST
  cont>   AS SELECT * FROM SALARY_HISTORY
  cont>           WHERE SALARY_AMOUNT > 60000
  cont>   WITH CHECK OPTION CONSTRAINT TEST_VIEW_CONST;
  SQL>
  SQL> INSERT INTO TEST (SALARY_AMOUNT) VALUES (50);
  %RDB-E-INTEG_FAIL, violation of constraint TEST_VIEW_CONST-
  caused operation to fail
  ```

- When you insert or update a view, the rows are stored in the base tables. If you do not use the WITH CHECK OPTION clause, you can insert or update rows through a view that do not conform to the view's definition. Once stored, however, you cannot retrieve those rows through the view because they do not meet the conditions specified by the view definition.

  For instance, the following view definition allows only salaries over $60,000. However, you can name the view in an INSERT statement to store a salary value of $50, which you can then retrieve only by referring to the table on which the view is based.

```
SQL> CREATE VIEW TEMP
cont>   AS SELECT * FROM SALARY_HISTORY
cont>           WHERE SALARY_AMOUNT > 60000;
SQL>
SQL> INSERT INTO TEMP (SALARY_AMOUNT) VALUES (50);
1 row inserted
SQL> -- Cannot get the row just stored through the view TEMP:
SQL> --
SQL> SELECT * FROM TEMP WHERE SALARY_AMOUNT < 100;
0 rows inserted
SQL> -- To retrieve the row, select it from the base table
SQL> --
SQL> SELECT * FROM SALARY_HISTORY WHERE SALARY_AMOUNT < 100;
 EMPLOYEE_ID   SALARY_AMOUNT   SALARY_START             SALARY_END
 NULL                  50.00   NULL                     NULL
1 row inserted
```

- The CREATE VIEW statement fails when both of the following are true:
  - The database to which it applies was created with the DICTIONARY IS REQUIRED argument.
  - The database was attached using the FILENAME argument.

  Under these circumstances, the statement fails with the following error when you issue it:

  ```
  %RDB-E-NO_META_UPDATE, metadata update failed
  -RDMS-F-CDDISREQ, CDD required for metadata updates
                    is not being maintained
  ```

- You can create up to 53,247 views. These values are architectural limits restricted by the on-disk structure. When you exceed the maximum limit for views, Oracle Rdb issues the MAXVIEWID error message.

  Views can have a record ID that ranges from 12288 through 65535.

  If you delete older views, Oracle Rdb recycles their identifiers so that the CREATE VIEW statement can succeed even after reaching the maximum value.

## Examples

Example 1: Defining a view based on a single table

This example shows a view definition that uses three columns from a single table, EMPLOYEES.

```
SQL> CREATE VIEW EMP_NAME
cont>   AS SELECT
cont>           FIRST_NAME,
cont>           MIDDLE_INITIAL,
cont>           LAST_NAME
cont>      FROM EMPLOYEES;
SQL> --
SQL> -- Now display the rows from the view just created.
SQL> SELECT * FROM EMP_NAME;
 FIRST_NAME   MIDDLE_INITIAL   LAST_NAME
 Alvin        A                Toliver
 Terry        D                Smith
               .
               .
               .
```

Example 2: Defining a view that does not allow you to insert or update rows that do not conform to the view's definition

This example shows a view definition using the WITH CHECK OPTION clause.

```
SQL>  CREATE VIEW ADMN_VIEW
cont>       AS SELECT * FROM JOB_HISTORY
cont>       WHERE DEPARTMENT_CODE = 'ADMN'
cont>       WITH CHECK OPTION CONSTRAINT ADMN_VIEW_CONST;
SQL> -- You cannot insert a row that does not
SQL> -- conform to the view definition.
SQL> --
SQL>  INSERT INTO ADMN_VIEW (DEPARTMENT_CODE) VALUES
('MBMN');
%RDB-E-INTEG-FAIL, violation of constraint ADMN_VIEW_CONST-
caused operation to fail
```

Example 3: Defining a view based on multiple tables

You can also define a view using more than one table.

```
SQL> CREATE VIEW CURRENT_SALARY
cont>  AS SELECT
cont>       E.LAST_NAME,
cont>       E.FIRST_NAME,
cont>       E.EMPLOYEE_ID,
cont>       SH.SALARY_START,
cont>       SH.SALARY_AMOUNT
cont>     FROM
cont>       SALARY_HISTORY SH, EMPLOYEES E
cont>     WHERE
cont>       SH.EMPLOYEE_ID = E.EMPLOYEE_ID
cont>     AND
cont>       SH.SALARY_END IS NULL           ;
```

This example defines a view from the EMPLOYEES and SALARY_HISTORY tables. It uses the select expression to:

- Choose the columns derived from each table. Because no column names are specified before the select expression, the columns inherit the names from the source tables.
- Join the tables and limit the view to current salaries.

Example 4: Defining a view with local column names

```
SQL> CREATE VIEW EMP_JOB
cont>      ( CURRENT_ID,
cont>        CURRENT_NAME,
cont>        CURRENT_JOB,
cont>        SUPERVISOR   )
cont>  AS SELECT
cont>        E.EMPLOYEE_ID,
cont>        E.LAST_NAME,
cont>        J.JOB_TITLE,
cont>        JH.SUPERVISOR_ID
cont>     FROM
cont>        EMPLOYEES E,
cont>        JOB_HISTORY JH,
cont>        JOBS J
cont>     WHERE
cont>        E.EMPLOYEE_ID = JH.EMPLOYEE_ID
cont>     AND
cont>        JH.JOB_CODE = J.JOB_CODE
cont>     AND
cont>        JH.JOB_END IS NULL      ;
```

This view definition:

- Specifies local names for the columns in the view.
- Joins the EMPLOYEES and JOB_HISTORY tables. This join links rows in the EMPLOYEES table to rows in the JOB_HISTORY table.
- Joins the JOB_HISTORY and JOBS tables. This join lets the view contain job titles instead of job codes.
- Uses the JH.JOB_END IS NULL expression. This clause specifies that only the current JOB_HISTORY rows, where the JOB_END column is null, should be included in the view.

The following query uses the view defined in the previous example:

```
EXEC SQL
        DECLARE X CURSOR FOR
        SELECT  CURRENT_ID, CURRENT_NAME, CURRENT_JOB, SUPERVISOR
        FROM    EMP_JOB
END-EXEC

EXEC SQL
        OPEN X
END-EXEC


PERFORM WHILE SQLCODE NOT = 0

        EXEC SQL
                FETCH X
                INTO :ID, :NAME, :JOB, :SUPER
        END-EXEC

END PERFORM


EXEC SQL
        CLOSE X
END-EXEC
```

Example 5: Defining a view with a calculated column

This example shows a view definition that derives a column through a calculation based on a column in an base table.

```
SQL> CREATE VIEW SS_DEDUCTION
cont>       ( IDENT,
cont>         SALARY,
cont>         SS_AMOUNT )
cont>  AS SELECT
cont>       E.EMPLOYEE_ID,
cont>       SH.SALARY_AMOUNT,
cont>       SH.SALARY_AMOUNT * 0.065
cont>     FROM
cont>       SALARY_HISTORY SH, EMPLOYEES E
cont>     WHERE
cont>       SH.EMPLOYEE_ID = E.EMPLOYEE_ID
cont>     AND
cont>       SH.SALARY_END IS NULL   ;
```

Each time this view definition executes, it computes a new virtual column, called SS_AMOUNT, from the SALARY_AMOUNT column of the SALARY_HISTORY table.

Example 6: Defining a view dependent on another view

This example creates a view, DEPENDENT_VIEW, that refers to the CURRENT_JOB view in its definition to include current job information for employees in the engineering department. Although you can define views that refer to other views, performance improves when you define view definitions that refer only to base tables.

```
SQL> CREATE VIEW DEPENDENT_VIEW
cont>   AS SELECT * FROM CURRENT_JOB
cont>          WHERE DEPARTMENT_CODE = 'ENG';
```

## DECLARE ALIAS Statement

Specifies the name and the source of the database definitions to be used for module compilation, and makes the named alias part of the implicit environment of an application. You can name either a file or a repository path name (on OpenVMS only) to be used for the database definitions.

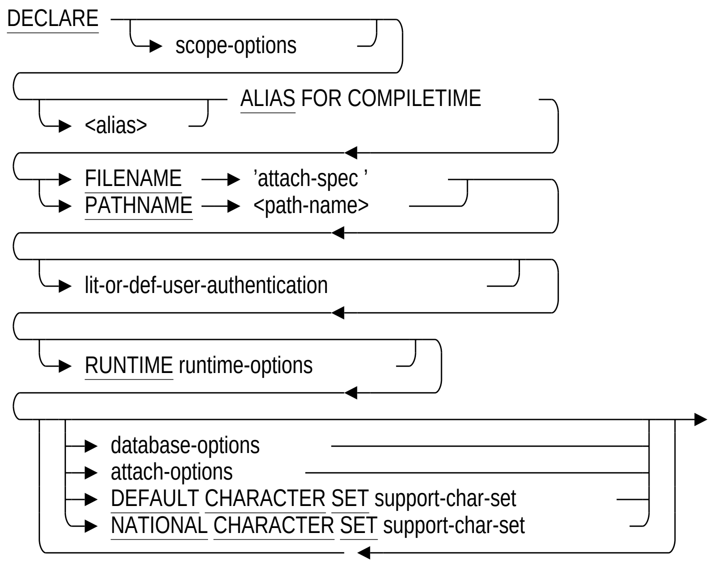
### Environment

You can use the DECLARE ALIAS statement:

- Embedded in host language programs to be precompiled
- In a context file
- As part of the DECLARE section in an SQL module

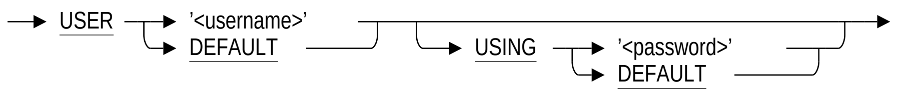
The alias that you declare must be different from any other alias specified in the module.

### Format

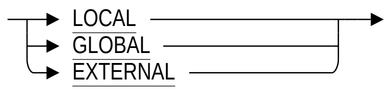


## DECLARE ALIAS Statement

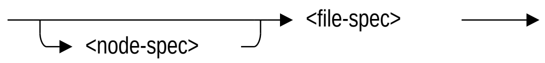
lit-or-def-user-authentication =



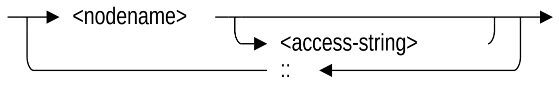
scope-options =



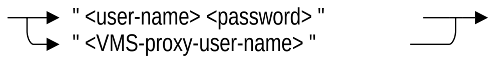
attach-spec =



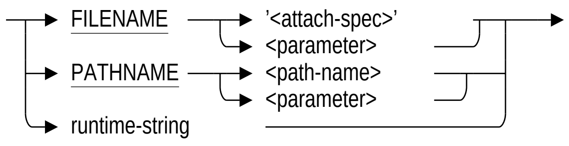
node-spec =



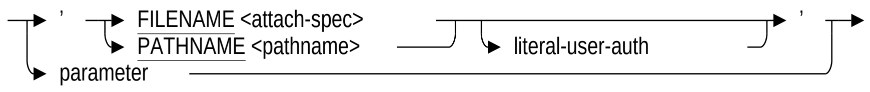
access-string =



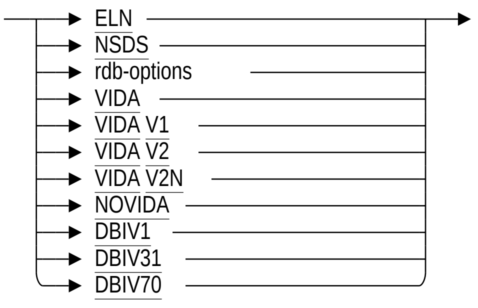
runtime-options =



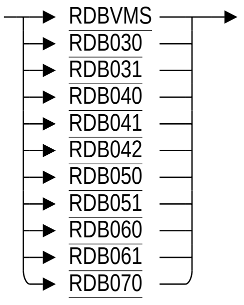
runtime-string =



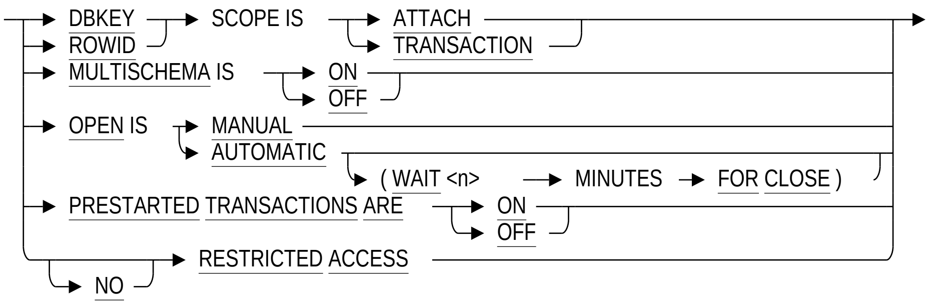
database-options =



rdb-options =



attach-options =



## Arguments

### scope-options

Specifies the scope of the database declaration in programs or module language procedures.

**LOCAL**
**GLOBAL**
**EXTERNAL**
Specifies the scope of the alias declaration in precompiled SQL or SQL module language.

The scope-option declarations are:

- LOCAL declares an alias that is local to procedures in the module in which it is declared, or local to dynamic statements prepared in the module in which it is declared.

  SQL attaches to a database with LOCAL scope only when you execute a procedure in the same module without a session. The alias of a database with LOCAL scope pertains only to that module.

  If the execution of a procedure in another module has attached to the implicit environment and that procedure subsequently calls another procedure that references a local database, SQL attempts to attach to that local database. If no transaction is active, SQL adds the local database to the implicit environment for this module. If a transaction is active, SQL returns an error message.
- GLOBAL declares an alias definition that is global to procedures in the application. GLOBAL is the default.
- EXTERNAL declares an external reference to a global alias that is defined in another module.

In single-image OpenVMS applications, the distinction between alias definitions and alias references is often unimportant. It is only necessary that each alias have at least one definition. For this reason, Oracle Rdb has treated all alias references (declared with the EXTERNAL keyword) the same as alias definitions (declared with the GLOBAL keyword or the default.) For compatibility with previous versions, this remains the default.

However, OpenVMS applications that share aliases between multiple images and Digital UNIX applications require a distinction between alias definitions and alias references. All definitions of any aliases shared between multiple OpenVMS images must be defined in one image, generally the shareable image against which you link the other images. Digital UNIX applications require one and only one definition for each alias, with any number of references to each alias.

Oracle Rdb recommends that you distinquish alias definitions from alias references in any new source code. Use the GLOBAL (or default) scope keyword for alias definitions and the EXTERNAL keyword for alias references. If you share aliases between multiple OpenVMS images, use the NOEXTERNAL_GLOBALS command line qualifier to override the default and cause SQL to properly treat alias references as references.

If you use the EXTERNAL_GLOBAL or the –`extern` command line qualifier, SQL treats aliases declared with the EXTERNAL keyword as GLOBAL. That is, SQL initializes alias references as well as alias definitions.

If you use the NOEXTERNAL_GLOBAL or the –`noextern` command line qualifier, SQL treats aliases declared with the EXTERNAL keyword as alias references and does not initialize them. It initializes all other aliases.

OpenVMS VAX OpenVMS Alpha On OpenVMS, the EXTERNAL_GLOBAL qualifier is the default. ♦

Digital UNIX On Digital UNIX, the –`noextern` option is the default. ♦

The [NO]INITIALIZE_HANDLES or the –`[no]init` command line qualifiers also affect the initialization of aliases, but they are recommended only for use in versions previous to V7.0.

See Section 3.5, Section 3.6, Section 4.3, and Section 4.4 for more information about the command line qualifiers.

**alias ALIAS**
Specifies a name for the attach to the database. Specifying an alias lets your program or interactive SQL statements refer to more than one database.

You do not have to specify an alias in the DECLARE ALIAS statement. The default alias in interactive SQL and in precompiled programs is RDB$DBHANDLE. In the SQL module language, the default is the alias specified in the module header. Using the default alias (either by specifying it explicitly in the DECLARE ALIAS statement or by omitting any alias) makes the database part of the default environment. Specifying a default database means that statements that refer to the default database do not need to use an alias.

If a default alias was already declared and you specify the default alias in the alias clause (or specify any alias that was already declared), you receive an error when you precompile the program or process it with the SQL module processor.

**FOR COMPILETIME**
Optional keyword provided for upward compatibility: DECLARE ALIAS specifies the compile-time environment by default. Specifies that the alias declared is the source of the database definition for program compiling and execution.

**FILENAME 'attach-spec'**
A quoted string containing full or partial information needed to access a database.

For an Oracle Rdb database, an attach specification contains the file specification of the .rdb file.

When you use the FILENAME argument, any changes you make to database definitions are entered *only* to the database system file, not to the repository. If you specify FILENAME, your application attaches to the database with that file name at run time.

If you specify FILENAME:

- During compilation, your application attaches to the specified database and reads metadata from the database definitions.
- At run time, your application attaches to the specified database.

For information regarding node-spec and file-spec, see Section 2.2.1.1.

OpenVMS VAX OpenVMS Alpha

**PATHNAME path-name**
A full or relative repository path name that specifies the source of the database definitions. When you use the PATHNAME argument, any changes you make to database definitions are entered in both the repository and the database system file. Oracle Rdb recommends using the PATHNAME argument if you have the repository on your system and you plan to use any data definition statements.

If you specify PATHNAME:

- During compilation, your application attaches to the repository database definition and reads metadata from the dictionary definitions. SQL extracts the file name of the Oracle Rdb database from the dictionary and saves it for use at run time.
- At run time, your application attaches to the Oracle Rdb database file name extracted from the dictionary at compilation.

The PATHNAME clause is available only on OpenVMS platforms. ♦

**lit-or-def-user-authentication**
Specifies the user name and password to enable access to databases, particularly remote databases.

You can use this clause to explicitly provide user name and password information in the DECLARE ALIAS statement.

**USER 'username'**
**USER DEFAULT**
Specifies the operating system user name that the database system uses for privilege checking.

You can specify a character string literal for the user name or you can specify the DEFAULT keyword. The DEFAULT keyword allows you to avoid placing the user name in a program's source code. If you specify the DEFAULT keyword, you pass the user name to the program by using a command line qualifier when you compile an SQL module or precompiled program. You use the USERNAME qualifier on OpenVMS and the `user` option on Digital UNIX.

**USING 'password'**
**USING DEFAULT**
Specifies the user's password for the user name specified in the USER clause.

You can specify a character string literal for the PASSWORD or you can specify the DEFAULT keyword. The DEFAULT keyword allows you to avoid placing the user name in a program's source code. If you specify the DEFAULT keyword, you pass the password to the program by using a command line qualifier when you compile an SQL module or precompiled program. You use the PASSWORD qualifier on OpenVMS and the `pass` option on Digital UNIX.

**RUNTIME runtime-options**
Specifies the source of the database definitions when the program is run.

**FILENAME 'attach-spec'**
**FILENAME parameter**
A quoted string containing full or partial information needed to access a database at run time or a parameter to hold the string.

For an Oracle Rdb database, the string or parameter contains the file specification of the .rdb file.

OpenVMS VAX OpenVMS Alpha

**PATHNAME path-name**
A string or parameter that holds the full or relative repository path name that specifies the source of the database definitions to be used at run time. When you use the PATHNAME argument, any changes you make to database definitions are entered in both the repository and the database system file.

The PATHNAME clause is available only on OpenVMS platforms. ♦

**runtime-string**
A quoted string or parameter that specifies the file name or path name of the database to be accessed at run time, and optionally, the user name and password of the user accessing the database at run time.

**literal-user-auth**
Specifies the user name and password for the specified database to be accessed at run time. For more information about when to use this clause, see the ATTACH Statement.

**USER 'username'**
A character string literal that specifies the operating system user name that the database system uses for privilege checking.

**USING 'password'**
A character string literal that specifies the user's password for the user name specified in the USER clause.

**database-options**
By default, SQL uses only the database options used to compile a program as valid options for that program. If you want to use the program with other supported databases, you can override the default options by specifying database options in the ATTACH or DECLARE ALIAS statement.

For more information on database options, see Section 2.10.

**DBKEY SCOPE IS ATTACH**
**DBKEY SCOPE IS TRANSACTION**
Controls when the database key of an erased record may be used again by SQL. There are two options for the DBKEY SCOPE clause.

- The default DBKEY SCOPE IS TRANSACTION clause means that SQL can reuse the database key of a deleted table row (to refer to a newly inserted row) as soon as the transaction that deleted the original row completes with a COMMIT statement. (If the user who deleted the original row enters a ROLLBACK statement, then the database key for that row cannot be used again by SQL.)

During the connection of the user or program who issued the DECLARE ALIAS statement, the DBKEY SCOPE IS TRANSACTION clause specifies that a database key is guaranteed to refer to the same row *only* within a particular transaction.

**Note**

Oracle Rdb recommends using DBKEY SCOPE IS TRANSACTION to reclaim space on a database page faster than if you use DBKEY SCOPE IS ATTACH.

---

- The DBKEY SCOPE IS ATTACH clause means that SQL cannot use the database key again (to refer to a newly inserted row) until the user who deleted the original row detaches from the database, unless another user is attached using DBKEY SCOPE IS ATTACH. (You detach by declaring another database with the same alias or by using the DISCONNECT statement.)

  During the connection of the user or program that issued the DECLARE ALIAS statement, the DBKEY SCOPE IS ATTACH argument specifies that a database key is guaranteed to refer to the same row until the user detaches from the database.

  With the DBKEY SCOPE IS ATTACH clause, a user or program can complete one or several transactions and, while still attached to the database, use database keys (obtained through INSERT, DECLARE CURSOR, FETCH, or singleton SELECT statements), to directly access table rows with less locking and greater speed.

If one user is connected to the database in DBKEY SCOPE IS ATTACH mode, all users are forced to operate in this mode, even if they are are explicitly connected in TRANSACTION mode. That is, no one reuses dbkeys until the ATTACH session disconnects.

See Section 2.6.5 for more information.

**ROWID SCOPE IS ATTACH**
**ROWID SCOPE IS TRANSACTION**
The ROWID keyword is a synonym for the DBKEY keyword. See the DBKEY SCOPE IS argument earlier in this Arguments list for more information.

**MULTISCHEMA IS ON**
**MULTISCHEMA IS OFF**
The MULTISCHEMA IS ON clause enables multischema naming for the duration of the database attach. The MULTISCHEMA IS OFF clause disables

multischema naming for the duration of the database attach. Multischema naming is disabled by default.

**PRESTARTED TRANSACTIONS ARE ON**
**PRESTARTED TRANSACTIONS ARE OFF**
Specifies whether Oracle Rdb enables or disables prestarted transactions.

Use the PRESTARTED TRANSACTIONS ARE OFF clause only if your application uses a server process that is attached to the database for long periods of time and causes the snapshot file to grow excessively. If you use the PRESTARTED TRANSACTIONS ARE OFF clause, Oracle Rdb uses additional I/O because each SET TRANSACTION statement must reserve a transaction sequence number (TSN).

For most applications, Oracle Rdb recommends that you enable prestarted transactions. The default is PRESTARTED TRANSACTIONS ARE ON. If you use the PRESTARTED TRANSACTIONS ARE ON clause or do not specify the PRESTARTED TRANSACTIONS clause, the COMMIT or ROLLBACK statement for the previous read/write transaction automatically reserves the TSN for the next transaction and reduces I/O.

You can define the RDMS$BIND_PRESTART_TXN logical name or the RDB_BIND_PRESTART_TXN configuration parameter to define the default setting for prestarted transactions outside of an application. The PRESTARTED TRANSACTION clause overrides this logical name or configuration parameter. For more information, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

**RESTRICTED ACCESS**
**NO RESTRICTED ACCESS**
Restricts access to the database. This allows you to access the database but locks out all other users until you disconnect from the database. Setting restricted access to the database requires DBADM privileges.

The default is NO RESTRICTED ACCESS if not specified.

**DEFAULT CHARACTER SET support-char-set**
Specifies the default character set of the alias at compile time. For a list of allowable character set names, see Section 2.1.

**NATIONAL CHARACTER SET support-char-set**
Specifies the national character set of the alias at compile time. For a list of allowable character set names, see Section 2.1.

## Usage Notes

- DECLARE ALIAS is a nonexecutable statement that declares the database to the program at compilation. SQL does not attach to the database until it executes the first executable SQL statement in the program or SQL module.
- When SQL executes the first procedure in a module, by default it attaches to each alias in the module that is active.
- In interactive or dynamic SQL, you must use the ATTACH statement to add a database to the implicit environment. For more information, see the ATTACH Statement.
- The DECLARE ALIAS statements embedded in programs or in the DECLARE section of an SQL module must come before any DECLARE TRANSACTION or executable SQL statements. The DECLARE ALIAS statements tell the application what databases it can compile against.
- To use an alias with a multischema database, you must enable ANSI/ISO quoting and create a delimited identifier, as described in Section 2.2.3.
- You must ensure that the character sets specified by the DEFAULT CHARACTER SET and NATIONAL CHARACTER SET clauses are the same as the actual character sets of the database that is accessed at run time. If these character sets do not match, unexpected results occur at run time.
- The default character set specifies the character set for columns with CHAR and VARCHAR data types. For more information on the default character set, see Section 2.1.3.
- A national character set specifies the character set for columns with the NCHAR and NCHAR VARYING data types. For more information on the national character set, see Section 2.1.4.
- If the default character set is not specified in the DECLARE ALIAS statement, the default character set of the database file invoked at compile time is assumed.
- If the national character set is not specified in the DECLARE ALIAS statement, the national character set of the database file invoked at compile time is assumed.

OpenVMS VAX OpenVMS Alpha

- If the database default character set is not DEC_MCS, the PATHNAME specifier cannot be used due to a current limitation of the repository where object names must only contain DEC_MCS characters. SQL flags this as an error. ♦

## Examples

Example 1: Specifying a database and an alias in embedded SQL

This statement declares the database defined by the file specification personnel. The precompiler uses this definition when compiling the program and SQL uses the file personnel.rdb when the program runs.

```
EXEC SQL
       DECLARE PERS_ALIAS ALIAS FOR FILENAME personnel
END-EXEC
```

Example 2: Specifying a database with restricted access

This statement is the same as Example 1, but specifies restricted access to the database.

```
EXEC SQL
       DECLARE PERS_ALIAS ALIAS FOR FILENAME personnel
       RESTRICTED ACCESS
END-EXEC
```

Example 3: Specifying the DECLARE ALIAS statement

This portion of an application program declares the databases MIA1 and MIA_CHAR_SET. The precompiler uses the MIA1 database when compiling the program and SQL uses the MIA_CHAR_SET database when the program runs.

```
EXEC SQL
         DECLARE ALIAS
      COMPILETIME FILENAME MIA1
             RUNTIME FILENAME MIA_CHAR_SET
             DEFAULT CHARACTER SET DEC_KANJI
             NATIONAL CHARACTER SET KANJI;
```

Example 4: Specifying the DEFAULT user authentication

The following example shows how to use the DEFAULT clause for user name and password in an SQL module:

```
MODULE              TEST_DECLARE
DIALECT             SQL92
LANGUAGE            C
PARAMETER           COLONS
ALIAS               RDB$DBHANDLE
-------------------------------------------------------
----------------------declarations-------------------
    DECLARE ALIAS COMPILETIME FILENAME mf_personnel
                  USER DEFAULT
                  USING DEFAULT
    RUNTIME :run_time_spec
   .
   .
   .
```

You pass the compile-time user name and password to the program by using command line qualifiers. For example, to compile the program on Digital UNIX, use the following command line:

```
$ sqlmod TESTDEC -user heleng -pass helenspasswd
```

At run time, the host language program can prompt the run-time user to specify only the file specification or the file specification and the user name and password at run time. The host language program can build the run time string.

For example, if the host language program uses only the file specification, the value of the variable passed to the program can be the following:

```
FILENAME "mf_personnel"
```

If the host language program uses the file specification, user name and password, the value of the variable passed to the program can be the following:

```
FILENAME "mf_personnel 'USER heleng' USING 'mypassword' "
```

You must enclose the string in quotation marks; whether you use single (') or double quotation marks (") depends upon the programming language.

If you use the following DECLARE ALIAS statement, the host language program can only prompt the run-time user to specify the file name.

```
    DECLARE ALIAS COMPILETIME FILENAME mf_personnel
                  USER DEFAULT
                  USING DEFAULT
    RUNTIME FILENAME :foo
```

## DECLARE CURSOR Statement

Declares a cursor.

With cursors, the conditions that define the result table are specified by the select expression in the DECLARE CURSOR statement. SQL creates the result table when it executes an OPEN statement. The result table for a cursor exists until a CLOSE, COMMIT, or ROLLBACK statement executes, the program stops, or you exit from interactive SQL. However, the result table can exist across transactions if you define a **holdable cursor**. A holdable cursor can remain open and retain its position when a new SQL transaction begins.

Host language programs require cursors because programs must perform operations one row or element at a time, and therefore may execute statements more than once to process an entire result table or list.

The **scope** of a cursor describes the portion of a module or program where the cursor is valid. The **extent** of a cursor tells how long it is valid. All cursors in SQL have the scope of the entire module.

You can create three classes of cursors, depending on which DECLARE CURSOR statement you use:

- The DECLARE CURSOR statement is executed immediately. A cursor that you create with this statement, sometimes called a **static** cursor, exists only within the scope and extent of its module. Both the cursor name and SELECT statement are known to your application at compile time.
- The dynamic DECLARE CURSOR statement is executed immediately. The cursor name is known at compile time, and the SELECT statement is determined at run time. You must supply a name for the SELECT statement that is generated at run time. A dynamic cursor exists within the scope of its module, but its extent is the entire run of the program or image. For information about the dynamic DECLARE CURSOR statement, see the DECLARE CURSOR Statement, Dynamic.
- The extended dynamic DECLARE CURSOR statement must be precompiled or used as part of a procedure in an SQL module. You must supply parameters for the cursor name and for the identifier of a prepared SELECT statement that is generated at run time. An extended dynamic cursor exists within the scope and extent of the entire module. For information about the extended dynamic DECLARE CURSOR statement, see the DECLARE CURSOR Statement, Extended Dynamic.

Within each class, you can create two types of cursors:

- **Table cursors** are a method that SQL provides to access individual rows of a result table. (A **result table** is a temporary collection of columns and rows from one or more tables or views.)
- **List cursors** are a method that SQL provides to access individual elements in a list.

  A **list** is an ordered collection of elements, or segments, of the data type LIST OF BYTE VARYING. For more information about the LIST OF BYTE VARYING data type, see Section 2.3.6.

  List cursors enable users to scan through a very large data structure from within a language that does not provide support for objects of such size. Because lists exist as a set of elements within a row of a table, a list cursor must refer to a table cursor because the table cursor provides the row context.

Cursors are further divided according to the modes of operations that they can perform. Table cursors have four modes:

- **Update** cursors are the default table cursor. Rows are first read and locked for SHARED READ or PROTECTED READ and then later, when an UPDATE is performed, the rows are locked for EXCLUSIVE access. If the table is reserved for EXCLUSIVE access, the original read lock is not required.
- **Read-only** cursors can be used to access row information from a result table whenever you do not intend to update the database. For example, you could use a read-only cursor to fetch row and column information for display.
- **Insert-only** cursors position themselves on a row that has just been inserted so that you can load lists into that row.
- **Update-only** cursors are used whenever you intend to modify many rows in the result table. When the UPDATE ONLY option is used, SQL uses a more aggressive lock mode that locks the rows for EXCLUSIVE access when first read. This mode avoids a lock promotion from SHARED READ or PROTECTED READ to EXCLUSIVE access. It may, therefore, avoid deadlocks normally encountered during the lock promotion.

List cursors have two modes:

- **Read-only** cursors are the default list cursor. They enable you to read existing lists. By adding the SCROLL keyword to the read-only list cursor clause, you enable Oracle Rdb to scroll forward and backward through the list segments as needed.
- **Insert-only** cursors enable you to insert data into a list.

Table 6–6 lists the classes, types, and modes of cursors that SQL provides.

**Table 6–6 Classes, Types, and Modes of Cursors**

| DECLARE CURSOR | | Dynamic DECLARE CURSOR | | Extended Dynamic DECLARE CURSOR | |
|---|---|---|---|---|---|
| **Table** | **List** | **Table** | **List** | **Table** | **List** |
| Insert-only | Insert-only | Insert-only | Insert-only | Insert-only | Insert-only |
| Read-only | Read-only | Read-only | Read-only | Read-only | Read-only |
| Update-only | | Update-only | | Update-only | |

For example, you must declare an insert-only table cursor to insert data into a table. If the table includes lists, use the table cursor to position on the correct row, and declare an insert-only list cursor to load the lists into that row. For details about using cursors to load data into your database, see the INSERT Statement.

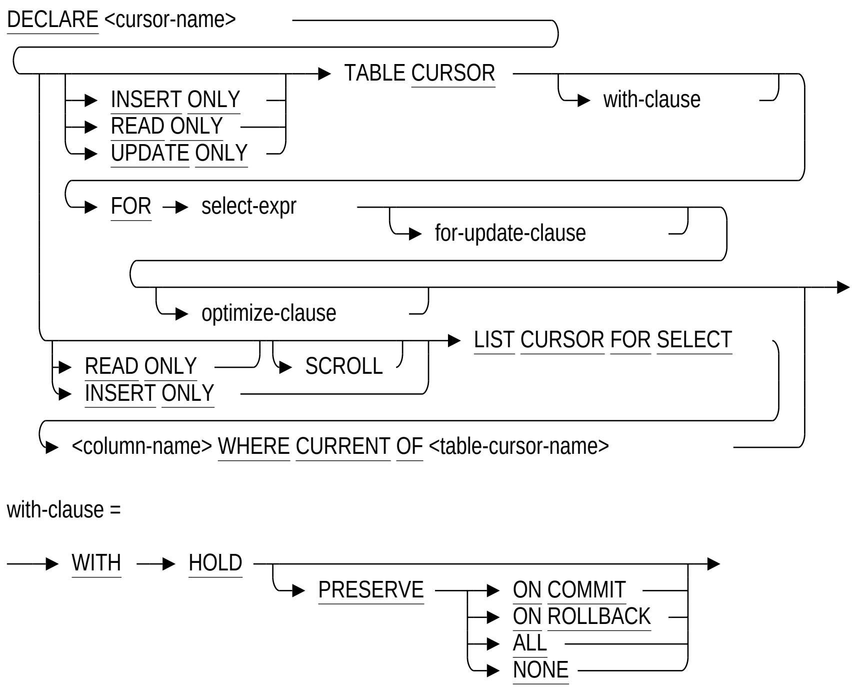
To process the rows of a result table formed by a DECLARE CURSOR statement, you must use the OPEN statement to position the cursor before the first row. Subsequent FETCH statements retrieve the values of each row for display on the terminal or processing in a program. (You must close the cursor before you attempt to reopen it.) You can similarly process the elements of a list by using an OPEN statement to position the cursor before the first element in the list and repeating FETCH statements to retrieve successive elements.

## Environment

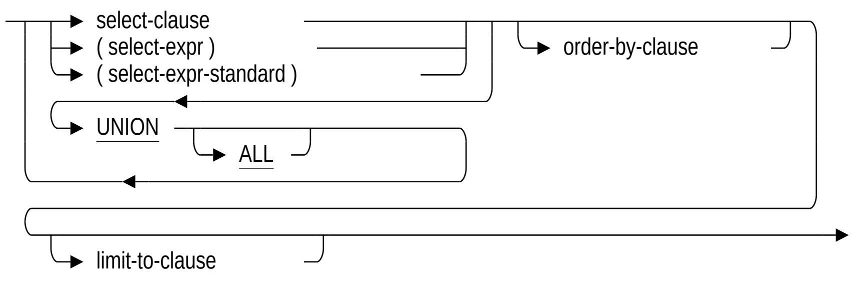
You can use the DECLARE CURSOR statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of the DECLARE section in an SQL module
- In a context file

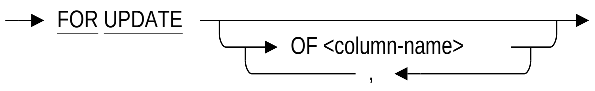
## Format



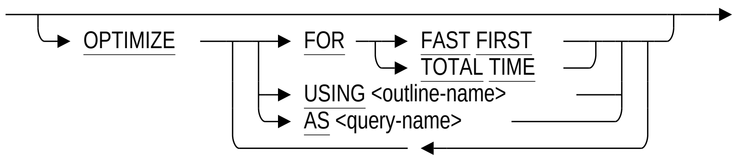
select-expr =



for-update-clause =



optimize-clause =



## Arguments

**cursor-name**

Specifies the name of the cursor you want to declare. Use a name that is unique among all the cursor names in the module. Use any valid SQL name. See Section 2.2 for more information on user-supplied names.

You can use a parameter to specify the cursor name at run time in an extended dynamic DECLARE CURSOR statement. See the DECLARE CURSOR Statement, Extended Dynamic for more information on the extended dynamic DECLARE CURSOR statement.

**INSERT ONLY**

Specifies that a new list or a new row is created or opened.

If you specify a list cursor but do not specify the INSERT ONLY clause, SQL declares a read-only list cursor by default.

If you specify a table cursor but do not specify the INSERT ONLY clause, SQL declares an update cursor by default.

When you specify an insert-only cursor, all the value expressions in the select list must be read/write. When you declare an insert-only table cursor to insert lists, you must specify both table column and list column names in the FROM clause.

For more information about how to use insert-only cursors, see the INSERT Statement.

#### READ ONLY

Specifies that the cursor is not used to update the database.

#### UPDATE ONLY

Specifies that the cursor is used to update the database.

Use an update-only cursor when you plan to update most of the rows you are fetching. The update-only cursor causes Oracle Rdb to apply more restrictive locking during the initial read operation, so that locks do not need to be upgraded later from READ to exclusive WRITE. This reduces the total number of lock requests per query, and may help to avoid deadlocks.

Use update-only table cursors to modify table rows. SQL does not allow update-only list cursors.

#### TABLE CURSOR

Specifies that the cursor you want to declare is a table cursor, rather than a list cursor. If you do not specify a cursor type, SQL declares a table cursor by default.

#### WITH HOLD

Indicates that the cursor remain open and maintain its position after the transaction ends. This is called a **holdable cursor**.

The WITH HOLD and WITH HOLD PRESERVE ON COMMIT syntax is synonymous. On commit, all cursors close except those defined with the WITH HOLD clause. On rollback, all cursors close including those defined with the WITH HOLD clause.

If you do not specify the WITH HOLD clause, the default behavior is to close all cursors on close, commit, or rollback when the program stops or when you exit from interactive SQL. This is synonymous to specifying the WITH HOLD PRESERVE NONE clause.

**PRESERVE ON COMMIT**
**PRESERVE ON ROLLBACK**
**PRESERVE ALL**
**PRESERVE NONE**
Specifies when a cursor remains open.

- PRESERVE ON COMMIT

  On commit, all cursors close except those defined with the WITH HOLD PRESERVE ON COMMIT syntax. On rollback, all cursors close including those defined with the WITH HOLD PRESERVE ON COMMIT syntax.

  This is the same as specifying the WITH HOLD clause without any preserve options.

- PRESERVE ON ROLLBACK

  On rollback, all cursors close except those defined with the WITH HOLD PRESERVE ON ROLLBACK syntax. On commit, all cursors close including those defined with the WITH HOLD PRESERVE ON ROLLBACK syntax.

- PRESERVE ALL

  All cursors remain open after commit or rollback. Cursors close with the CLOSE statement or when the session ends.

- PRESERVE NONE

  All cursors close after a CLOSE, COMMIT, or ROLLBACK statement, when the program stops, or when you exit from interactive SQL.

  This is the same as not specifying the WITH HOLD clause at all.

**FOR select-expr**
A select expression that defines which columns and rows of which tables SQL includes in the cursor. See Section 2.8.1 for more information on select expressions.

**FOR UPDATE OF column-name**
Specifies the columns in a cursor that you or your program might later modify with an UPDATE statement. The column names in the FOR UPDATE clause must belong to a table or view named in the FROM clause.

You do not have to specify the FOR UPDATE clause of the DECLARE CURSOR statement to later modify rows using the UPDATE statement:

- If you do specify a FOR UPDATE clause and later specify columns in the UPDATE statement that are not in the FOR UPDATE clause, SQL issues a warning message and proceeds with the update modifications.

- If you do not specify a FOR UPDATE clause, you can update any column using the UPDATE statement. SQL does not issue any messages.

If you have set your dialect to ORACLE LEVEL1, the FOR UPDATE OF clause in a SELECT statement provides UPDATE ONLY CURSOR semantics by locking all the rows selected.

**OPTIMIZE FOR**

The OPTIMIZE FOR clause specifies the preferred optimizer strategy for statements that specify a select expression. The following options are available:

- FAST FIRST

  A query optimized for FAST FIRST returns data to the user as quickly as possible, even at the expense of total throughput.

  If a query can be cancelled prematurely, you should specify FAST FIRST optimization. A good candidate for FAST FIRST optimization is an interactive application that displays groups of records to the user, where the user has the option of aborting the query after the first few screens. For example, singleton SELECT statements default to FAST FIRST optimization.

  If optimization strategy is not explicitly set, FAST FIRST is the default.

- TOTAL TIME

  If your application runs in batch, accesses all the records in the query, and performs updates or writes a report, you should specify TOTAL TIME optimization. Most queries benefit from TOTAL TIME optimization.

  The following examples illustrate the DECLARE CURSOR syntax for setting a preferred optimization mode:

```
SQL> DECLARE TEMP1 TABLE CURSOR
cont>  FOR
cont>    SELECT *
cont>      FROM EMPLOYEES
cont>      WHERE EMPLOYEE_ID > '00400'
cont>  OPTIMIZE FOR FAST FIRST;
SQL> --
SQL> DECLARE TEMP2 TABLE CURSOR
cont>  FOR
cont>    SELECT LAST_NAME, FIRST_NAME
cont>      FROM EMPLOYEES
cont>        ORDER BY LAST_NAME
cont>  OPTIMIZE FOR TOTAL TIME;
```

**OPTIMIZE USING outline-name**

Explicitly names the query outline to be used with the select expression even if the outline IDs for the select expression and for the outline are different.

See the CREATE OUTLINE Statement for more information on creating an outline.

**OPTIMIZE AS query-name**
Assigns a name to the query. You must define the RDMS$DEBUG_FLAGS logical name or the RDB_DEBUG_FLAGS configuration parameter be the letter *S* to see the access methods used to produce the results of the query.

**SCROLL**
Specifies that Oracle Rdb can read the items in a list from either direction (up or down) or at random. The SCROLL keyword must be used if the following fetch options are desired:

- NEXT
- PRIOR
- FIRST
- LAST
- RELATIVE
- ABSOLUTE

If SCROLL is not specified, the default for FETCH is NEXT.

**LIST CURSOR**
Specifies a cursor that is used to manipulate columns of the data type LIST OF BYTE VARYING.

**FOR SELECT column-name**
Specifies a column of data type LIST OF BYTE VARYING.

**WHERE CURRENT OF table-cursor-name**
Specifies the table cursor that provides the row context for the list cursor. The table cursor named must be defined using a DECLARE CURSOR statement.

## Usage Notes

- You refer to cursors in INSERT, OPEN, CLOSE, FETCH, UPDATE, and DELETE statements. The order of those statements in a host language source file is not important; a CLOSE statement for a cursor can precede its corresponding OPEN statement so long as program control branches to process the OPEN statement first at run time. However, you must close a cursor before you reopen it.

- You can use the SQL CLOSE statement to close cursors individually, or use the sql_close_cursors( ) routine to close all open cursors. The sql_close_cursors( ) routine takes no arguments. For an example of this routine, see the *Oracle Rdb7 Guide to Distributed Transactions*.
- SQL does not restrict how many cursors you can have open at once. It is valid to declare and open more than one cursor at a time. However, if you plan to use static, dynamic, and extended dynamic cursors within the same program, you should avoid giving the same name to different cursors that share the same scope or extent.
- You cannot refer to list cursors in UPDATE or DELETE statements.
- SQL considers as read-only cursors those that:
  - Use the DISTINCT argument to eliminate duplicate rows from the result table
  - Name more than one table or view in the FROM clause
  - Include an aggregate function in the select list
  - Contain a GROUP BY or HAVING clause

  When a cursor is declared as READ ONLY, it can never be referenced in a positional UPDATE or DELETE statement or an INSERT INTO cursor-name statement.

  When a cursor has neither INSERT ONLY, READ ONLY, or UPDATE ONLY specified, it is considered a general cursor that can be used for a DELETE, INSERT or UPDATE statement. However, if any of the above listed items occurs, SQL implicitly considers the cursor to be a READ ONLY cursor.
- You can process a table cursor only in the forward direction. If you want to move the table cursor back to a row that you already processed, you must close the table cursor and open it again.
- The order of the result table is unpredictable unless you specify an ORDER BY clause in the DECLARE CURSOR statement. (The ORDER BY clause is not valid in a list cursor declaration.)
- SQL evaluates the result table of the cursor (specified by the SELECT statement) when it executes an OPEN statement for the cursor.
- SQL evaluates any parameters in the select expression of a DECLARE CURSOR statement when it executes the OPEN statement for the cursor. It cannot evaluate the parameters again until you close and open the cursor again.

- If a DECLARE CURSOR statement contains parameters, you pass the parameters to it by declaring them in the procedure that contains the OPEN statement. In addition, you must specify the parameter in the host language call to the procedure that contains the OPEN statement. Because the DECLARE CURSOR statement appears in the declaration section of a module, not a procedure, you cannot pass the parameters directly to the DECLARE CURSOR statement.

  For examples of declaring cursors with parameters and passing parameters to an SQL module, see Chapter 3.

- You cannot refer to insert-only cursors in the following statements:
  - DELETE and UPDATE statements that specify the CURRENT OF clause
  - FETCH statements
- You cannot use the INSERT ONLY clause in a DECLARE CURSOR statement that contains one or more of the following clauses:
  - DISTINCT
  - WHERE
  - ORDER BY
  - GROUP BY
  - UNION
- You can use only an insert-only cursor for the cursor name in an INSERT statement used to add a new row to a table cursor or a new element to a list cursor.
- When you define an insert-only table cursor, you must include the LIST column in the select list of the table cursor. For an example, see Example 3.
- A DECLARE CURSOR statement that uses parameters to specify statements and cursor names is an extended dynamic DECLARE CURSOR statement. An extended dynamic DECLARE CURSOR statement lets programs supply cursor and statement names at run time. See the DECLARE CURSOR Statement, Extended Dynamic for more information on the extended dynamic DECLARE CURSOR statement.

  An extended dynamic DECLARE CURSOR statement is an executable statement and returns a status value. In the module language, you must include such a statement in a procedure.

- When accessing SQL segmented strings, you must be careful to close a list cursor before you fetch the next row in the table cursor. If you fetch some, but not all, rows from a list cursor and move to the next row in the table cursor without closing the list cursor, you continue to fetch rows from the previous list cursor. SQL does not issue a warning or error message telling you that you opened two list cursors.

```
SQL> -- Define a cursor of Board Manufacturing Department Managers:
SQL> --
SQL> DECLARE BM_MGR CURSOR FOR
cont> SELECT EMPLOYEE_ID, RESUME FROM RESUMES R, CURRENT_INFO CI
cont> WHERE R.EMPLOYEE_ID = CI.ID AND DEPARTMENT
cont> CONTAINING "BOARD MANUFACTURING" AND JOB = "Department Manager";
SQL> --
SQL> -- Define a cursor for resumes of those managers:
SQL> DECLARE THE_RESUME LIST CURSOR FOR
cont> SELECT RESUME WHERE CURRENT OF BM_MGR;
SQL> --
SQL> -- Build the manager’s cursor:
SQL> OPEN BM_MGR;
SQL> --
SQL> -- Fetch the manager’s row:
SQL> FETCH BM_MGR;
 R.EMPLOYEE_ID   R.RESUME
 00164                          72:2:3
SQL> --
SQL> -- Get part of the resume:
SQL> OPEN THE_RESUME;
SQL> FETCH THE_RESUME;
 RESUME
 This is the resume for Alvin Toliver
SQL> --
SQL> -- Do not close the resume, and access the next manager:
SQL> FETCH BM_MGR;
 R.EMPLOYEE_ID   R.RESUME
 00166                          72:2:9
SQL> -- SQL continues to fetch from Toliver’s resume (00164)
SQL> -- because the list cursor was not closed.
SQL> -- If it were a new resume, you would see
SQL> -- a new "This is the resume for ..." line.
SQL> FETCH THE_RESUME;
 RESUME
 Boston, MA
```

- The declared cursor must refer to the same table or list of tables specified in a SET TRANSACTION RESERVING table-name statement. For example:

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION RESERVING jobs FOR WRITE;
SQL> DECLARE curs1 CURSOR WITH HOLD FOR
cont> SELECT first_name,last_name FROM employees;
SQL> OPEN CURS1;
%RDB-E-UNRES_REL, relation EMPLOYEES in specified request is not a relation
reserved in specified transaction
```

- You can specify only the WITH HOLD clause for table cursors.
- The data stored in the temporary area created by the cursor may be obsolete. For example, user BROWN declares and opens a cursor accessing the employees table. User JONES deletes an employee from the employees table during the time BROWN has the cursor open. BROWN still sees the employee deleted by JONES because BROWN is accessing a temporary area containing the original table and, now obsolete, data.
- You can define an SQL session default setting for holdable cursors using the SET HOLD CURSORS statement. See the SET HOLD CURSORS Statement for more information.
- The WITH HOLD PRESERVE ALL clause conforms to the ODBC driver behavior of cursors.
- If an outline exists, Oracle Rdb uses the outline specified in the OPTIMIZE USING clause unless one or more of the directives in the outline cannot be followed. For example, if the compliance level for the outline is mandatory and one of the indexes specified in the outline directives has been deleted, the outline is not used. SQL issues an error message if an existing outline cannot be used.

  If you specify the name of an outline that does not exist, Oracle Rdb compiles the query, ignores the outline name, and searches for an existing outline with the same outline ID as the query. If an outline with the same outline ID is found, Oracle Rdb attempts to execute the query using the directives in that outline. If an outline with the same outline ID is not found, the optimizer selects a strategy for the query for execution.

  See the *Oracle Rdb7 Guide to Database Performance and Tuning* for more information regarding query outlines.

## Examples

Example 1: Declaring a table cursor in interactive SQL

The following example declares a cursor named SALARY_INFO. The result table for SALARY_INFO contains the names and current salaries of employees and is sorted by last name.

```
SQL> --
SQL> DECLARE SALARY_INFO CURSOR FOR
cont>  SELECT  E.FIRST_NAME, E.LAST_NAME, S.SALARY_AMOUNT
cont>  FROM    EMPLOYEES E, SALARY_HISTORY S
cont>  WHERE   E.EMPLOYEE_ID = S.EMPLOYEE_ID
cont>          AND
cont>          S.SALARY_END IS NULL
cont>  ORDER BY
cont>          E.LAST_NAME ASC;
SQL> --
SQL> -- Use an OPEN statement to open the cursor and
SQL> -- position it before the first row of the
SQL> -- result table:
SQL> OPEN SALARY_INFO;
SQL> --
SQL> -- Finally, use two FETCH statements to see the
SQL> -- first two rows of the cursor:
SQL> FETCH SALARY_INFO;
 E.FIRST_NAME   E.LAST_NAME      S.SALARY_AMOUNT
 Louie          Ames                  $26,743.00
SQL> FETCH SALARY_INFO;
 E.FIRST_NAME   E.LAST_NAME      S.SALARY_AMOUNT
 Leslie         Andriola              $50,424.00
```

OpenVMS VAX OpenVMS Alpha

Example 2: Declaring a table cursor in a PL/I program

This program fragment uses embedded DECLARE CURSOR, OPEN, and FETCH statements to retrieve and print the names and departments of managers.

```
/* Declare the cursor: */
EXEC SQL DECLARE MANAGER CURSOR FOR
        SELECT E.FIRST_NAME, E.LAST_NAME, D.DEPARTMENT_NAME
                FROM EMPLOYEES E, DEPARTMENTS D
                WHERE E.EMPLOYEE_ID = D.MANAGER_ID ;

/* Open the cursor: */
EXEC SQL OPEN MANAGER;
```

```
/* Start a loop to process the rows of the cursor: */
DO WHILE (SQLCODE = 0);
/* Retrieve the rows of the cursor
and put the value in host language variables: */
EXEC SQL FETCH MANAGER INTO :FNAME, :LNAME, :DNAME;
/* Print the values in the variables: */
   .
   .
   .
END;

/* Close the cursor: */
EXEC SQL CLOSE MANAGER;
♦
```

Example 3: Using table and list cursors to retrieve list data in interactive SQL

The following example declares a table and list cursor to retrieve list information:

```
SQL> DECLARE TBLCURSOR INSERT ONLY TABLE CURSOR FOR
cont> SELECT EMPLOYEE_ID, RESUME FROM RESUMES;
SQL> DECLARE LSTCURSOR INSERT ONLY LIST CURSOR FOR SELECT RESUME WHERE
CURRENT OF TBLCURSOR;
SQL> OPEN TBLCURSOR;
SQL> INSERT INTO CURSOR TBLCURSOR (EMPLOYEE_ID) VALUES ('00164');
1 row inserted
SQL> OPEN LSTCURSOR;
SQL> INSERT INTO CURSOR LSTCURSOR VALUES ('This is the resume for 00164');
SQL> INSERT INTO CURSOR LSTCURSOR VALUES ('Boston, MA');
SQL> INSERT INTO CURSOR LSTCURSOR VALUES ('Oracle Corporation');
SQL> CLOSE LSTCURSOR;
SQL> CLOSE TBLCURSOR;
SQL> COMMIT;
SQL> DECLARE TBLCURSOR2 CURSOR FOR SELECT EMPLOYEE_ID,
cont> RESUME FROM RESUMES;
SQL> DECLARE LSTCURSOR2 LIST CURSOR FOR SELECT RESUME WHERE
CURRENT OF TBLCURSOR2;
```

```
SQL> OPEN TBLCURSOR2;
SQL> FETCH TBLCURSOR2;
  00164
SQL> OPEN LSTCURSOR2;
SQL> FETCH LSTCURSOR2;
 RESUME
 This is the resume for 00164
SQL> FETCH LSTCURSOR2;
 RESUME
 Boston, MA
SQL> FETCH LSTCURSOR2;
 RESUME
 Oracle Corporation
SQL> FETCH LSTCURSOR2;
 RESUME
%RDB-E-STREAM_EOF, attempt to fetch past end of record stream
SQL> CLOSE LSTCURSOR2;
SQL> SELECT * FROM RESUMES;
%SQL-W-PRINT_SSID, Segmented string id for RESUME will be displayed
  EMPLOYEE_ID    RESUME
 00164                    1:701:2
1 row selected
SQL> CLOSE TBLCURSOR2;
SQL> COMMIT;
```

Example 4: Using the scroll attribute for a list cursor

The following example declares a table and read-only scrollable list cursor to retrieve list information by scrolling back and forth between segments of the list:

```
SQL> DECLARE CURSOR_ONE
cont>   TABLE CURSOR FOR
cont>   (SELECT EMPLOYEE_ID,RESUME FROM RESUMES);
SQL> --
SQL> DECLARE CURSOR_TWO
cont>    READ ONLY
cont>    SCROLL
cont>    LIST CURSOR
cont>    FOR SELECT RESUME
cont>    WHERE CURRENT OF CURSOR_ONE;
```

Example 5: Declaring a holdable cursor

```
SQL> -- Declare a holdable cursor that remains open on COMMIT
SQL> --
SQL> DECLARE curs1 CURSOR WITH HOLD PRESERVE ON COMMIT FOR
cont> SELECT e.first_name,e.last_name FROM employees e;
SQL> OPEN curs1;
SQL> FETCH curs1;
 FIRST_NAME   LAST_NAME
 Terry        Smith
SQL> FETCH curs1;
 FIRST_NAME   LAST_NAME
 Rick         O’Sullivan
SQL> COMMIT;
SQL> FETCH curs1;
 FIRST_NAME   LAST_NAME
 Stan         Lasch
SQL> FETCH curs1;
 FIRST_NAME   LAST_NAME
 Susan        Gray
SQL> ROLLBACK;
SQL> FETCH curs1;
%SQL-F-CURNOTOPE, Cursor CURS1 is not opened
SQL> --
SQL> -- Declare another holdable cursor that remains open on ROLLBACK
SQL> --
SQL> DECLARE curs2 CURSOR WITH HOLD PRESERVE ON ROLLBACK FOR
cont> SELECT e.first_name,e.last_name FROM employees e;
SQL> OPEN curs2;
SQL> FETCH curs2;
 FIRST_NAME   LAST_NAME
 Terry        Smith
SQL> FETCH curs2;
 FIRST_NAME   LAST_NAME
 Rick         O’Sullivan
SQL> ROLLBACK;
SQL> FETCH curs2;
 FIRST_NAME   LAST_NAME
 Stan         Lasch
SQL> FETCH curs2;
 FIRST_NAME   LAST_NAME
 Susan        Gray
SQL> COMMIT;
SQL> FETCH curs2;
%SQL-F-CURNOTOPE, Cursor CURS2 is not opened
```

## DECLARE CURSOR Statement, Dynamic

Declares a cursor where the SELECT statement is supplied at run time in a parameter.

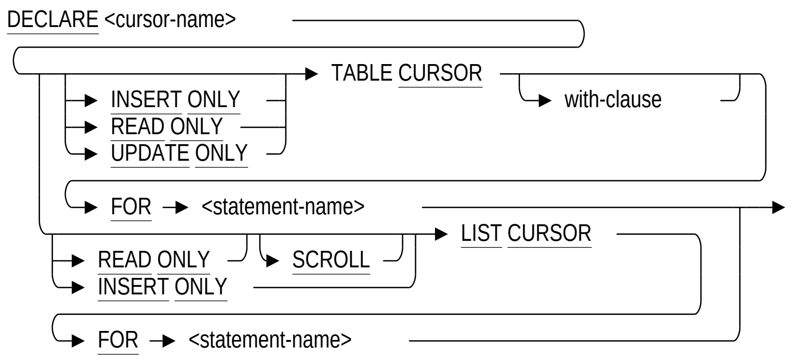
Refer to the DECLARE CURSOR Statement for a detailed description of statement elements that apply to both dynamic and nondynamic DECLARE CURSOR statements.

### Environment

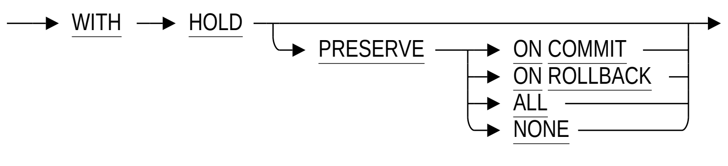
You can use the dynamic DECLARE CURSOR statement:

- Embedded in host language programs to be precompiled
- As part of the DECLARE statement section in an SQL module

### Format



with-clause =



## Arguments

**cursor-name**
The name of the cursor you want to declare. Use a name that is unique among all the cursor names in the module. Use any valid SQL name. See Section 2.2 for more information on identifiers.

**INSERT ONLY**
Specifies that a new list or a new row is created or opened.

**READ ONLY**
Specifies that the cursor is not used to update the database.

**UPDATE ONLY**
Specifies that the cursor is used to update the database.

**TABLE CURSOR**
Specifies that you are declaring a cursor to access the rows in a table.

**WITH HOLD**
Indicates that the cursor remain open and maintain its position after the transaction ends. This is called a **holdable cursor**.

The WITH HOLD and WITH HOLD PRESERVE ON COMMIT syntax is synonymous. On commit, all cursors close except those defined with the WITH HOLD clause. On rollback, all cursors close including those defined with the WITH HOLD clause.

If you do not specify the WITH HOLD clause, the default behavior is to close all cursors on close, commit, or rollback when the program stops or when you exit from interactive SQL. This is synonymous to specifying the WITH HOLD PRESERVE NONE clause.

**PRESERVE ON COMMIT**
**PRESERVE ON ROLLBACK**
**PRESERVE ALL**
**PRESERVE NONE**
Specifies when a cursor remains open.

- PRESERVE ON COMMIT

  On commit, all cursors close except those defined with the WITH HOLD PRESERVE ON COMMIT syntax. On rollback, all cursors close including those defined with the WITH HOLD PRESERVE ON COMMIT syntax.

  This is the same as specifying the WITH HOLD clause without any preserve options.

- PRESERVE ON ROLLBACK

  On rollback, all cursors close except those defined with the WITH HOLD PRESERVE ON ROLLBACK syntax. On commit, all cursors close including those defined with the WITH HOLD PRESERVE ON ROLLBACK syntax.
- PRESERVE ALL

  All cursors remain open after commit or rollback. Cursors close with the CLOSE statement or when the session ends.
- PRESERVE NONE

  All cursors close after a CLOSE, COMMIT, or ROLLBACK statement, when the program stops, or when you exit from interactive SQL.

  This is the same as not specifying the WITH HOLD clause at all.

**SCROLL**
Specifies that Oracle Rdb can read the items in a list from either direction (up or down) or at random.

**LIST CURSOR**
Specifies that you are declaring a cursor to access the elements in a list.

**FOR statement-name**
A name that identifies a prepared SELECT statement that is generated at run time.

## Usage Notes

- In a dynamic DECLARE CURSOR statement, the cursor name is compiled, but the SELECT statement is determined at run time.
- Because a dynamic DECLARE CURSOR statement is not executable, you must place this statement in the DECLARE section of an SQL module, as with static DECLARE CURSOR statements.
- Views that contain a GROUP BY or UNION clause in their definition cannot be accessed using dynamic cursors. In interactive SQL, these views can be accessed with the SELECT statement; in precompiled SQL or SQL module language, these views can be accessed with singleton SELECT statements and with nondynamic cursors. The problem shows up only with dynamic cursors. If a user attempts to access one of these views with a dynamic cursor, the following error is returned when the cursor is opened:

```
"RDMS-F-VIEWNORET, view cannot be retrieved by database key".
```

The workaround for this problem is to use nondynamic cursors to access the view. If a dynamic cursor must be used, the statement should access the base tables that make up the view (with the GROUP BY and UNION clauses, as appropriate) and not the view itself.

- The declared cursor must refer to the same table or list of tables specified in a SET TRANSACTION RESERVING table-name statement. For example:

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION RESERVING jobs FOR WRITE;
SQL> DECLARE curs1 CURSOR WITH HOLD FOR
cont> SELECT first_name,last_name FROM employees;
SQL> OPEN CURS1;
%RDB-E-UNRES_REL, relation EMPLOYEES in specified request is not a relation
reserved in specified transaction
```

- You can only specify the WITH HOLD clause for table cursors.
- Be aware that the data stored in the temporary area created by the cursor may be obsolete. For example, user BROWN declares and opens a cursor accessing the employees table. User JONES deletes an employee from the employees table during the time BROWN has the cursor open. BROWN still sees the employee deleted by JONES because BROWN is accessing a temporary area containing the original table and, now obsolete, data.
- The transaction setting must remain static during the time a holdable cursor is open.
- You can define a database default setting for holdable cursors using the SET HOLD CURSORS statement. See the SET HOLD CURSORS Statement for more information.
- The WITH HOLD PRESERVE ALL clause conforms to the ODBC driver behavior of cursors.

## Examples

Example 1: Using a parameter for a statement name

```
.
.
.
* This program prepares a statement for dynamic execution from the string
* passed to it, and uses a dynamic cursor to fetch a row from a table.
*
*/
#include <stdio.h>
#include <descrip.h>

struct SQLDA_STRUCT {
          char SQLDAID[8];
          int SQLDABC;
          short SQLN;
          short SQLD;
          struct {
            short SQLTYPE;
            short SQLLEN;
            char *SQLDATA;
            short *SQLIND;
            short SQLNAME_LEN;
            char SQLNAME[30];
              } SQLVAR[];
        } *SQLDA;
main()
{

/*
 *  General purpose locals
 */
int     i;
long    sqlcode;

char    command_string[256];


/*
 *  Allocate SQLDA structures.
 */

SQLDA = malloc(500);
SQLDA->SQLN = 20;

/* Get the SELECT statement at run time. */

printf("\n Enter a SELECT statement.\n");
printf("\n Do not end the statement with a semicolon.\n");
gets(command_string);
/* Prepare the SELECT statement. */
PREP_STMT( &sqlcode, &command_string, SQLDA );
if (sqlcode != 0)
    goto err;

/* Open the cursor. */
OPEN_CURSOR( &sqlcode );
if (sqlcode != 0)
    goto err;

/* Allocate memory.   */
for (i=0; i < SQLDA->SQLD; i++) {
    SQLDA->SQLVAR[i].SQLDATA = malloc( SQLDA->SQLVAR[i].SQLLEN );
    SQLDA->SQLVAR[i].SQLIND  = malloc( 2 );
}

/* Fetch a row. */
FETCH_CURSOR( &sqlcode, SQLDA );
if (sqlcode != 0)
    goto err;

/* Use the SQLDA to determine the data type of each column in the row
   and print the column.  For simplicity, test for only two data types.
   CHAR and INT. */

for (i=0; i < SQLDA->SQLD; i++) {

    switch (SQLDA->SQLVAR[i].SQLTYPE) {

    case SQLDA_CHAR;    /* Character */
        printf( "%s", SQLDA->SQLVAR[i].SQLDATA );
        break;
    case SQLDA_INTEGER:   /* Integer */
        printf( "%d", SQLDA->SQLVAR[i].SQLDATA );
        break;
    default:
        printf( "Some other datatype encountered\n");
    }
}

/* Close the cursor. */
CLOSE_CURSOR( &sqlcode );

ROLLBACK(&sqlcode );
return;
   .
   .
   .
}
```

Example 2: SQL module file that the preceding program calls

```
--  This program uses dynamic cursors to fetch a row.
--
--
MODULE          C_MOD_DYN_CURS
LANGUAGE        C
AUTHORIZATION   RDB$DBHANDLE

DECLARE ALIAS FOR FILENAME personnel

-- Declare the dynamic cursor. Use a statement name to identify a
-- prepared SELECT statement.

DECLARE CURSOR1 CURSOR FOR STMT_NAME

-- Prepare the statement from a statement entered at run time
-- and specify that SQL write information about the number and
-- data type of select list items to the SQLDA.

PROCEDURE PREP_STMT
    SQLCODE
    COMMAND_STRING  CHAR (256)
    SQLDA;

      PREPARE STMT_NAME SELECT LIST INTO SQLDA FROM COMMAND_STRING;


PROCEDURE OPEN_CURSOR
     SQLCODE;

    OPEN CURSOR1;

PROCEDURE FETCH_CURSOR
    SQLCODE
    SQLDA;

    FETCH CURSOR1 USING DESCRIPTOR SQLDA;

PROCEDURE CLOSE_CURSOR
    SQLCODE;

    CLOSE CURSOR1;

PROCEDURE ROLLBACK
    SQLCODE;

    ROLLBACK;
```

## DECLARE CURSOR Statement, Extended Dynamic

Declares an extended dynamic cursor. An extended dynamic DECLARE CURSOR statement is a DECLARE CURSOR statement in which both the cursor name and the SELECT statement are supplied in parameters at run time.

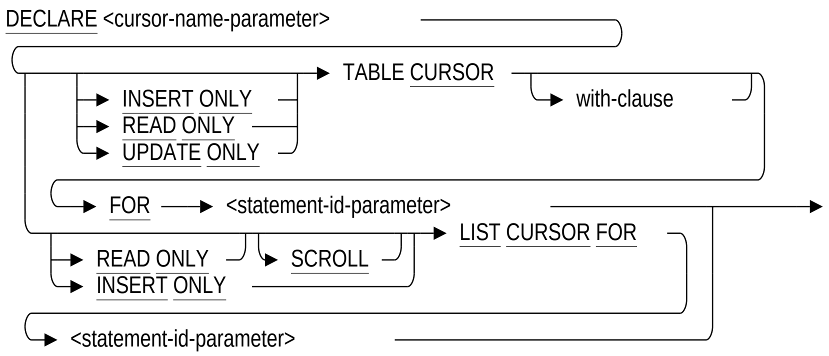
See the DECLARE CURSOR Statement for a detailed description of statement elements that apply to both dynamic and nondynamic DECLARE CURSOR statements.

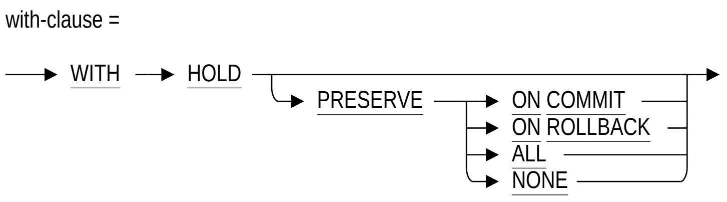
### Environment

You can use the extended dynamic DECLARE CURSOR statement:

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

### Format





## Arguments

**cursor-name-parameter**
Contains the name of the cursor you want to declare. Use a character string parameter to hold the cursor name that the program supplies at run time.

**INSERT ONLY**
Specifies that a new list or a new row is created or opened.

**READ ONLY**
Specifies that the cursor is not used to update the database.

**UPDATE ONLY**
Specifies that the cursor is used to update the database.

**TABLE CURSOR FOR**
Specifies that you are declaring a cursor to access the rows in a table.

**WITH HOLD**
Indicates that the cursor remain open and maintain its position after the transaction ends. This is called a **holdable cursor**.

The WITH HOLD and WITH HOLD PRESERVE ON COMMIT syntax is synonymous. On commit, all cursors close except those defined with the WITH HOLD clause. On rollback, all cursors close including those defined with the WITH HOLD clause.

If you do not specify the WITH HOLD clause, the default behavior is to close all cursors on close, commit, or rollback when the program stops or when you exit from interactive SQL. This is synonymous to specifying the WITH HOLD PRESERVE NONE clause.

**PRESERVE ON COMMIT**
**PRESERVE ON ROLLBACK**
**PRESERVE ALL**
**PRESERVE NONE**
Specifies when a cursor remains open.

- PRESERVE ON COMMIT

  On commit, all cursors close except those defined with the WITH HOLD PRESERVE ON COMMIT syntax. On rollback, all cursors close including those defined with the WITH HOLD PRESERVE ON COMMIT syntax.

  This is the same as specifying the WITH HOLD clause without any preserve options.

- PRESERVE ON ROLLBACK

  On rollback, all cursors close except those defined with the WITH HOLD PRESERVE ON ROLLBACK syntax. On commit, all cursors close including those defined with the WITH HOLD PRESERVE ON ROLLBACK syntax.
- PRESERVE ALL

  All cursors remain open after commit or rollback. Cursors close with the CLOSE statement or when the session ends.
- PRESERVE NONE

  All cursors close after a close, commit, or rollback statement, when the program stops, or when you exit from interactive SQL.

  This is the same as not specifying the WITH HOLD clause at all.

**FOR statement-id-parameter**
A parameter that contains an integer that identifies a prepared SELECT statement. Use an integer parameter to hold the statement identifier that SQL generates and assigns to the parameter when SQL executes a PREPARE statement.

**SCROLL**
Specifies that Oracle Rdb can read the items in a list from either direction (up or down) or at random.

**LIST CURSOR FOR**
Specifies that you are declaring a cursor to access the elements in a list.

## Usage Notes

- An extended dynamic DECLARE CURSOR statement is an executable statement in dynamic SQL. It lets you specify, through parameters, both the name of a cursor and the identifier of the SELECT statement on which the cursor is based at run time. In general, using extended dynamic SQL allows a single set of SQL procedures to concurrently control an arbitrary number of prepared statements as opposed to dynamic SQL where the control is sequential.
- The extended dynamic DECLARE CURSOR statement lets you use one DECLARE CURSOR-PREPARE statement combination for multiple, dynamically generated SELECT statements. This eliminates the necessity

of coding a DECLARE CURSOR and PREPARE statement for each dynamically generated SELECT statement.

- You must use parameters to specify both the cursor name and the statement identifier in an extended dynamic DECLARE CURSOR statement. Specifying either the cursor name or the statement identifier explicitly but not both through a parameter generates an error. Specifying both the cursor name and statement identifier explicitly makes the cursor a nondynamic cursor and the DECLARE CURSOR statement a nonexecutable statement.
- Because an extended dynamic DECLARE CURSOR statement is executable, it returns an execution status at run time. Your program should check the status after executing an extended dynamic DECLARE CURSOR statement.

  Because an extended dynamic DECLARE CURSOR statement is executable, you must place this statement in programs and SQL module files where executable statements are allowed. For example, you must place extended dynamic DECLARE CURSOR statements within a procedure in an SQL module, not in the DECLARE section as with static or dynamic DECLARE CURSOR statements.
- The declared cursor must refer to the same table or list of tables specified in a SET TRANSACTION RESERVING table-name statement. For example:

```
SQL> ATTACH 'FILENAME mf_personnel';
SQL> SET TRANSACTION RESERVING jobs FOR WRITE;
SQL> DECLARE curs1 CURSOR WITH HOLD FOR
cont> SELECT first_name,last_name FROM employees;
SQL> OPEN CURS1;
%RDB-E-UNRES_REL, relation EMPLOYEES in specified request is not a relation
reserved in specified transaction
```

- You can only specify the WITH HOLD clause for table cursors.
- Be aware that the data stored in the temporary area created by the cursor may be obsolete. For example, user BROWN declares and opens a cursor accessing the employees table. User JONES deletes an employee from the employees table during the time BROWN has the cursor open. BROWN still sees the employee deleted by JONES because BROWN is accessing a temporary area containing the original table and, now obsolete, data.
- The transaction setting must remain static during the time a holdable cursor is open.

- You can define a database default setting for holdable cursors using the SET HOLD CURSORS statement. See the SET HOLD CURSORS Statement for more information.
- The WITH HOLD PRESERVE ALL clause conforms to the ODBC driver behavior of cursors.

## Example

Example 1: Using parameters for statement and cursor names

The following example shows two procedures from the online sample program SQL$MULTI_STMT_DYN.SQLADA. These procedures show the use of parameters for statement and cursor names.

```
.
.
.
-- This procedure prepares a statement for dynamic execution from the string
-- passed to it.  This procedure can prepare any number of statements
-- because the statement is passed to it as the parameter, cur_procid.

procedure PREPARE_SQL is
    CUR_CURSOR : string(1..31) := (others => ' ');
    CUR_PROCID : integer := 0;
    CUR_STMT : string(1..1024) := (others => ' ');

begin
-- Allocate separate SQLDAs for parameter markers (sqlda_in) and select list
-- items (sqlda_out).  Assign the value of the constant MAXPARMS (set in the
-- declarations section) to the SQLN field of both SQLDA structures. SQLN
-- specifies to SQL the maximum size of the SQLDA.

sqlda_in := new sqlda_record;
sqlda_in.sqln := maxparms;
sqlda_out := new sqlda_record;
sqlda_out.sqln := maxparms;

-- Assign the SQL statement that was constructed in the procedure
-- CONSTRUCT_SQL to the variable cur_stmt.

cur_stmt := sql_stmt;

-- Use the PREPARE...SELECT LIST statement to prepare the dynamic statement
-- and write information about any select list items in it to sqlda_out.
-- It 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 SQLDA (specifically, the
-- sqlda_out SQLDA specified).
--
-- Note that the PREPARE statement could have prepared the statement without
-- writing to an SQLDA.  Instead, a separate DESCRIBE...SELECT LIST statement
-- would have written information about any select list items to an SQLDA.
```

```
EXEC SQL PREPARE :cur_procid SELECT LIST INTO :sqlda_out FROM :cur_stmt;
case sqlca.sqlcode is
    when sql_success => null;
    when others => raise syntax_error;
end case;

-- Use the DESCRIBE...MARKERS statement to write information about any
-- parameter markers in the dynamic statement to sqlda_in.  This statement
-- writes information to an SQLDA (specifically, the sqlda_in SQLDA
-- specified) about the number and data type of any parameter markers in
-- the prepared dynamic statement.  Note that SELECT statements may also
-- have parameter markers.

EXEC SQL DESCRIBE :cur_procid MARKERS INTO sqlda_in;
case sqlca.sqlcode is
    when sql_success => null;
    when others => raise syntax_error;
end case;

-- If the operation is "Read," create a unique name for the cursor name
-- so that the program can pass the cursor name to the dynamic DECLARE
-- CURSOR statement.

if cur_op(1) = 'R' then
    cur_cursor(1) := 'C';
    cur_cursor(2..name_strlng) := cur_name(1..name_strlng - 1);

-- Declare the dynamic cursor.

    EXEC SQL DECLARE :cur_cursor CURSOR FOR :cur_procid;
    case sqlca.sqlcode is
        when sql_success => null;
        when others => raise syntax_error;
    end case;
end if;

number_of_procs := number_of_procs + 1;
sqlda_in_array(number_of_procs) := sqlda_in;
sqlda_out_array(number_of_procs) := sqlda_out;
procedure_names(number_of_procs) := cur_name;
procedure_ids(number_of_procs) := cur_procid;
if cur_op(1) = 'R' then
    cursor_names(number_of_procs) := cur_cursor;
end if;
```

```
exception
    when syntax_error =>
        sql_get_error_text(get_error_buffer,get_error_length);
        put_line(get_error_buffer(1..integer(get_error_length)));
        put("Press RETURN to continue. ");
        get_line(terminal,release_screen,last);
        new_line;
end PREPARE_SQL;
.
.
.
begin  -- procedure body DISPLAY_DATA

-- Before displaying any data, allocate buffers to hold the data
-- returned by SQL.
--
    allocate_buffers;

-- Allocate and assign SQLDAs for the requested SQL procedure.
--
sqlda_in := new sqlda_record;
sqlda_in := sqlda_in_array(stmt_index);
sqlda_out := new sqlda_record;
sqlda_out := sqlda_out_array(stmt_index);
cur_cursor := cursor_names(stmt_index);
-- Open the previously declared cursor.  The statement specifies
-- an SQLDA (specifically, sqlda_in) as the source of addresses for any
-- parameter markers in the cursor's SELECT statement.
--
EXEC SQL OPEN :cur_cursor USING DESCRIPTOR sqlda_in;
case sqlca.sqlcode is
    when sql_success => null;
    when others => raise unexpected_error;
end case;

-- Fetch the first row from the result table.  This statement fetches a
-- row from the opened cursor and writes it to the addresses specified
-- in an SQLDA (specifically, sqlda_out).
--
EXEC SQL FETCH :cur_cursor USING DESCRIPTOR sqlda_out;
case sqlca.sqlcode is
--  Check to see if the result table has any rows.
    when sql_success => null;
    when stream_eof =>
        put_line("No records found.");
        new_line;
    when others => raise unexpected_error;
end case;

-- Set up a loop to display the first row, then fetch and display second
-- and subsequent rows.
```

```
    rowcount := 0;
    while sqlca.sqlcode = 0 loop
        rowcount := rowcount + 1;
--      Execute the DISPLAY_ROW procedure.
        display_row;
--      To only display 5 rows, exit the loop if the loop counter
--      equals MAXROW (coded as 5 in this program).
        if rowcount = maxrows then exit; end if;
--      Fetch another row, exit the loop if no more rows.
        EXEC SQL FETCH :cur_cursor USING DESCRIPTOR sqlda_out;
        case sqlca.sqlcode is
            when sql_success => null;
            when stream_eof => exit;
            when others => raise unexpected_error;
        end case;
    end loop;
-- Close the cursor.
EXEC SQL CLOSE :cur_cursor;
case sqlca.sqlcode is
    when sql_success => null;
    when others => raise unexpected_error;
end case;
exception
    when unexpected_error =>
        sql_get_error_text(get_error_buffer,get_error_length);
        EXEC SQL ROLLBACK;
        put_line("This condition was not expected.");
        put_line(get_error_buffer(1..integer(get_error_length)));
        put("Press RETURN to continue. ");
        get_line(terminal,release_screen,last);

-- Stop and let the user look before returning.
    skip;
    put_line("Press RETURN to proceed. ");
    get_line(terminal,release_screen,last);

end DISPLAY_DATA;
```

## DECLARE LOCAL TEMPORARY TABLE Statement

Explicitly declares a local temporary table.

The metadata for a declared local temporary table is not stored in the database and cannot be shared by other modules. These tables are sometimes called **scratch tables**.

The data stored in the table cannot be shared between SQL sessions or modules in a single session. Unlike persistent base tables, the metadata and data do not persist beyond an SQL session.

In addition to declared local temporary tables, there are two other types of temporary tables:

- Global temporary tables
- Local temporary tables

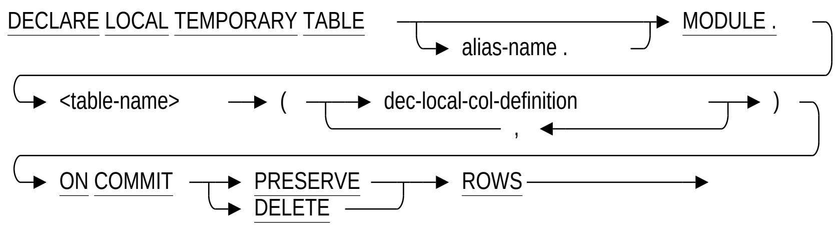
See the CREATE TABLE Statement for additional information on global and local temporary tables.

### Environment

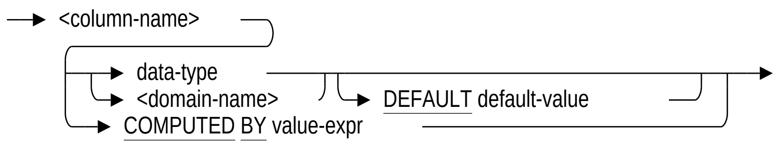
You can use the DECLARE LOCAL TEMPORARY TABLE statement:

- In interactive SQL
- In dynamic SQL as a statement to be dynamically executed
- In a stored module

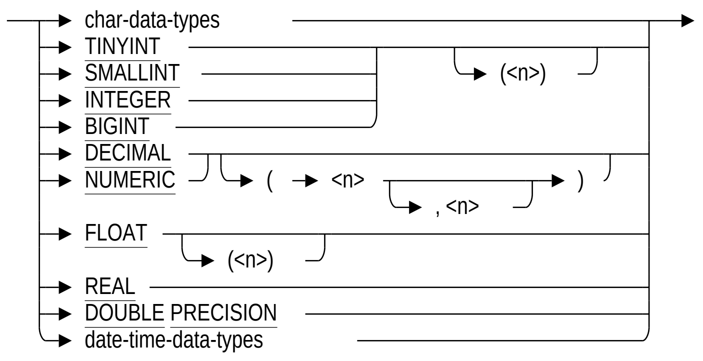
### Format



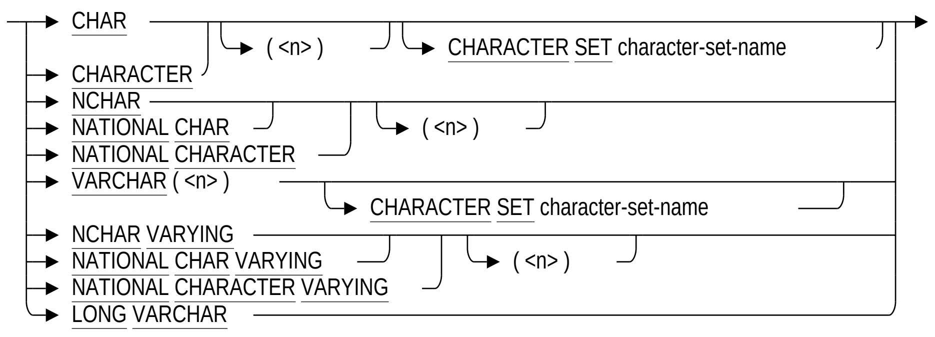
dec-local-col-definition =



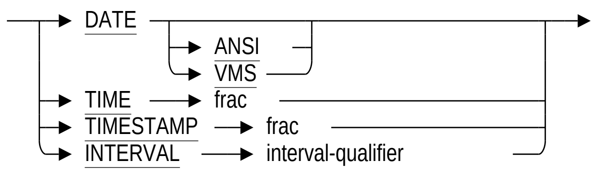
data-type =



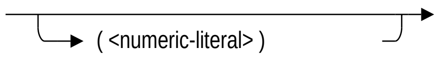
char-data-types =



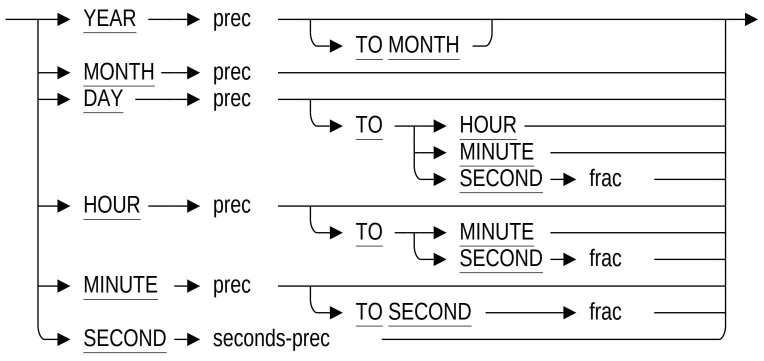
date-time-data-types =



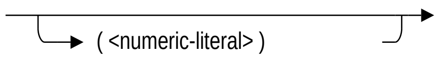
frac =



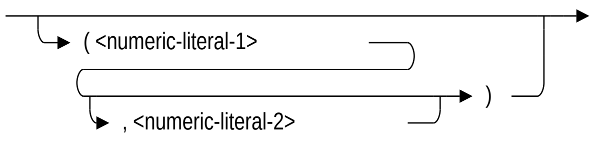
interval-qualifier =



prec =



seconds-prec =



## Arguments

**table-name**
The name of the table you want to declare. You can optionally precede the table-name with an alias-name and a period (.). You must, however, precede the table-name with the keyword MODULE and a period (.), for example, MODULE.EMPL_PAYROLL.

**dec-local-col-definition**
The definition for a column in the table. SQL gives you two ways to specify column definitions:

- By directly specifying a data type to associate with a column name
- By naming a domain that indirectly specifies a data type to associate with a column name

See the CREATE TABLE Statement for more information about column definitions. See Section 2.3 for more information about data types.

**ON COMMIT PRESERVE ROWS**
**ON COMMIT DELETE ROWS**
Specifies whether data is preserved or deleted after a COMMIT statement for declared local temporary tables.

The default, if not specified, is ON COMMIT DELETE ROWS.

## Usage Notes

- You must precede the name of the declared local temporary table with the keyword MODULE and a period (.), for example:

  ```
  SQL> DECLARE LOCAL TEMPORARY TABLE MODULE.empl_payroll
      .
      .
      .
  ```

- A declared local temporary table can have the same name as a persistent base table or a temporary table.
- Declared local temporary tables are stored in virtual memory, not in a storage area. They use the same storage segment layout as persistent base tables, but they use additional space in memory for management overhead. On OpenVMS, declared local temporary tables use 56 bytes per row for management overhead; on Digital UNIX, they use 88 bytes.

  See the *Oracle Rdb7 Guide to Database Design and Definition* for information on estimating the virtual memory needs of declared local temporary tables.
- Because the metadata is not stored in the database, you cannot use declared local temporary tables in as many places as you use persistent base tables. In particular, declared local temporary tables cannot:
  - Be deleted using the DROP TABLE statement

  - Be modified using the ALTER TABLE statement
  - Be truncated
  - Contain data of the data type LIST OF BYTE VARYING
  - Be referred to in a view or in a storage map
  - Be referred to in a constraint or be defined with a constraint
  - Contain indexes
  - Use triggers
  - Have granted or revoked privileges
  - Be referred to in an interactive or dynamic CREATE OUTLINE statement if the declared local temporary table is outside the defintion of a stored module
  - Be referred to in a COMMENT ON statement
  - Be specified in the RESERVING clause of a SET TRANSACTION statement
  - Be displayed using the SHOW statement
  - Be referenced in a COMPUTED BY column of another persistent or declared local temporary table
  - Be exported or imported
- You cannot define column or table constraints in declared local temporary tables. The columns in a declared local temporary table can reference domain constraints.
- You can use dbkeys with declared local temporary tables.
- Oracle Rdb does not journal changes to declared local temporary tables.
- You can define and write to a declared local temporary table during a read-only transaction.
- You can qualify the name of the table with an alias name. For example, if the database alias is PERS, the qualified name of PAYCHECK_DECL_TAB is PERS.MODULE.PAYCHECK_DECL_TAB. However, the declared local temporary table name is not an element of a catalog or schema.
- The following table summarizes the actions you can take using temporary tables and when you can refer to temporary tables.

| | Types of Temporary Tables | | |
|---|---|---|---|
| **Action** | **Global** | **Local** | **Declared Local** |
| Delete table | Yes | Yes | No |
| Modify table | No | No | No |
| Truncate table | Yes | No | No |
| Define constraints on table or column | Yes | No | No |
| Refer to table in constraint definition | Yes[2] | Yes | No |
| Refer to domain constraints | Yes | Yes | Yes |
| Refer to table in storage map | Yes | Yes | No |
| Refer to table in view | Yes | Yes | No |
| Grant privileges on temporary table | Yes | Yes | No |
| Refer to table in outline | Yes | Yes | No[1] |
| Define indexes on table | No | No | No |
| Use dbkeys on table | Yes | Yes | Yes |
| Use triggers with table | Yes | No | No |
| Refer to table in COMMENT ON statement | Yes | Yes | No |
| Contain LIST OF BYTE VARYING data | No | No | No |
| Specify in RESERVING clause | No | No | No |
| Write to table during read-only transaction | Yes | Yes | Yes |
| Create in a read-only transaction | No | No | Yes |
| Refer to a table in a computed-by column | Yes | Yes | No |

[1]You can refer to a declared local temporary table if it is defined inside a stored module.
[2]From a temporary table only.

For information about global and local temporary tables, see the CREATE TABLE Statement.

- Because the declared local temporary table name is qualified by the keyword MODULE and a period (.), a declared local temporary table can have the same name as a persistent base table or view.

## Examples

Example 1: Declaring and using a declared local temporary table in interactive SQL

```
SQL> DECLARE LOCAL TEMPORARY TABLE MODULE.PAYCHECK_DECL_INT
cont>        (EMPLOYEE_ID ID_DOM,
cont>         LAST_NAME CHAR(14),
cont>         HOURS_WORKED INTEGER,
cont>         HOURLY_SAL   INTEGER(2),
cont>         WEEKLY_PAY   INTEGER(2))
cont>        ON COMMIT PRESERVE ROWS;
SQL> --
SQL> INSERT INTO MODULE.PAYCHECK_DECL_INT
cont>          (EMPLOYEE_ID, LAST_NAME, HOURS_WORKED, HOURLY_SAL, WEEKLY_PAY)
cont>           SELECT P.EMPLOYEE_ID, E.LAST_NAME, P.HOURS_WORKED,
cont>                  P.HOURLY_SAL, P.HOURS_WORKED * P.HOURLY_SAL
cont>              FROM EMPLOYEES E, PAYROLL P
cont>                  WHERE E.EMPLOYEE_ID = P.EMPLOYEE_ID
cont>                    AND P.WEEK_DATE = DATE '1995-08-01';
100 rows inserted

SQL> SELECT * FROM MODULE.PAYCHECK_DECL_INT LIMIT TO 2 ROWS;
 EMPLOYEE_ID   LAST_NAME       HOURS_WORKED      HOURLY_SAL       WEEKLY_PAY
 00165         Smith                     40           30.50          1220.00
 00166         Dietrich                  40           36.00          1440.00
2 rows selected
```

Example 2: Creating a stored module that contains the following:

- A declared local temporary table, MODULE.PAYCHECK_DECL_TAB
- A procedure, PAYCHECK_INS_DECL, that inserts weekly salary records into the declared local temporary table, MODULE.PAYCHECK_DECL_TAB
- A procedure, LOW_HOURS_DECL, that counts the number of employees with less than 40 hours worked

The following example also demonstrates that you can access the declared local temporary table only from within the module.

```
SQL> -- Create the module containing a declared temporary table.
SQL> --
SQL> CREATE MODULE PAYCHECK_DECL_MOD
cont>   LANGUAGE SQL
cont>   DECLARE LOCAL TEMPORARY TABLE MODULE.PAYCHECK_DECL_TAB
cont>          (EMPLOYEE_ID ID_DOM,
cont>           LAST_NAME CHAR(14) ,
cont>           HOURS_WORKED INTEGER, HOURLY_SAL INTEGER(2),
cont>           WEEKLY_PAY   INTEGER(2))
cont>           ON COMMIT PRESERVE ROWS
cont> --
cont> -- Create the procedure to insert rows.
cont> --
cont>   PROCEDURE PAYCHECK_INS_DECL;
cont>   BEGIN
cont>     INSERT INTO MODULE.PAYCHECK_DECL_TAB
cont>          (EMPLOYEE_ID, LAST_NAME, HOURS_WORKED, HOURLY_SAL, WEEKLY_PAY)
cont>           SELECT P.EMPLOYEE_ID, E.LAST_NAME, P.HOURS_WORKED,
cont>                  P.HOURLY_SAL, P.HOURS_WORKED * P.HOURLY_SAL
cont>                  FROM EMPLOYEES E, PAYROLL P
cont>                  WHERE E.EMPLOYEE_ID = P.EMPLOYEE_ID
cont>                    AND P.WEEK_DATE = DATE '1995-08-01';
cont>   END;
cont> --
cont> -- Create the procedure to count the low hours.
cont> --
cont>   PROCEDURE LOW_HOURS_DECL (:cnt INTEGER);
cont>   BEGIN
cont>     SELECT COUNT(*) INTO :cnt FROM MODULE.PAYCHECK_DECL_TAB
cont>            WHERE HOURS_WORKED < 40;
cont>   END;
cont> END MODULE;
SQL> --
SQL> -- Call the procedure to insert the rows.
SQL> --
SQL> CALL PAYCHECK_INS_DECL();
SQL> --
SQL> -- Declare a variable and call the procedure to count records with
SQL> -- low hours.
SQL> --
SQL> DECLARE :low_hr_cnt integer;
SQL> CALL LOW_HOURS_DECL(:low_hr_cnt);
  LOW_HR_CNT
           2

SQL> --
SQL> -- Because the table is a declared local temporary table, you cannot
SQL> -- access it from outside the stored module that contains it.
SQL> --
SQL> SELECT * FROM MODULE.PAYCHECK_DECL_TAB;
%SQL-F-RELNOTDCL, Table PAYCHECK_DECL_TAB has not been declared in module or
environment
```

## DECLARE MODULE Statement

Specifies characteristics, such as character sets, quoting rules, and the default date format for a nonstored module.

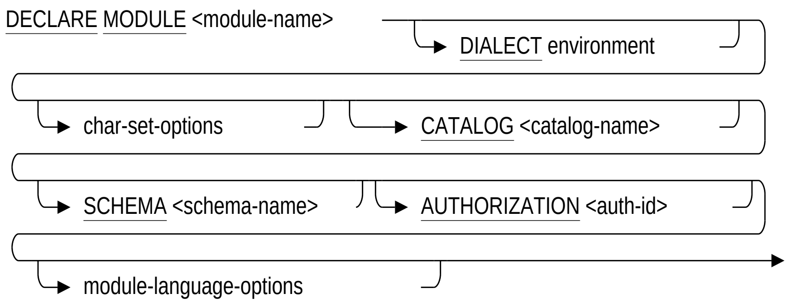
### Environment

You can use the DECLARE MODULE statement:

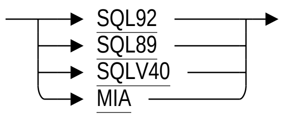
- Embedded in host language programs to be precompiled
- In a context file

This command is not executable.

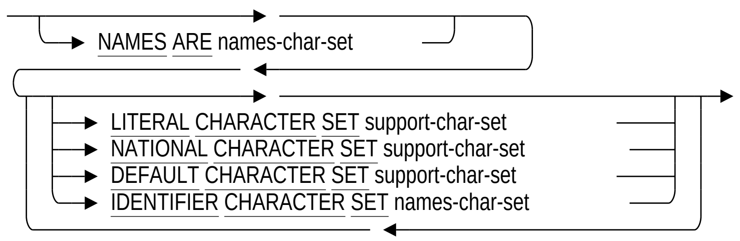
### Format



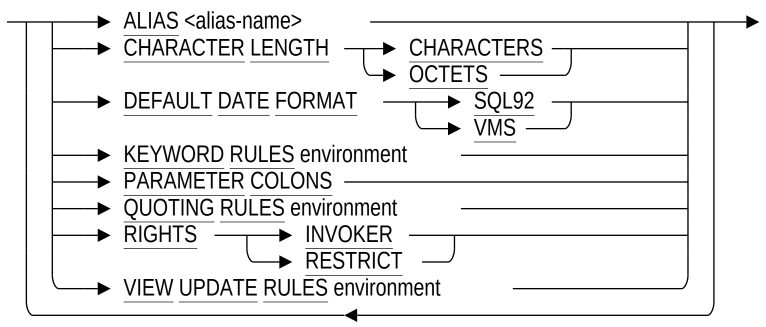
environment =



char-set-options =


NAMES ARE names-char-set
LITERAL CHARACTER SET support-char-set
NATIONAL CHARACTER SET support-char-set
DEFAULT CHARACTER SET support-char-set
IDENTIFIER CHARACTER SET names-char-set

module-language-options =



## Arguments

### MODULE module-name

An optional name for the nonstored module. If you do not supply a module name, the default name is SQL_MODULE.

Use any valid OpenVMS name. (See Section 2.2 for more information on user-supplied names.) However, the name must be unique among the modules that are linked together to form an executable image.

### DIALECT

Controls the following settings:

- Whether the length of character string parameters, columns, and domains are interpreted as characters or octets
- Whether double quotation marks are interpreted as string literals or delimited identifiers
- Whether or not identifiers can be keywords

- Which views are read-only
- Whether columns with the DATE or CURRENT_TIMESTAMP data type are interpreted as VMS or SQL92 format

The DIALECT clause lets you specify the settings with one clause, instead of specifying each setting individually. Because the module processor processes the module clauses sequentially, the DIALECT clause can override the settings of clauses specified before it or be overridden by clauses specified after it.

The following statements are specific to the SQL92 dialect:

- The default constraint evaluation time setting changes from DEFERRABLE to NOT DEFERRABLE.
- Conversions between character data types when storing data or retrieving data will raise exceptions or warnings in certain situations.
- You can specify DECIMAL or NUMERIC for formal parameters in SQL modules, and declare host language parameters with packed decimal or signed numeric storage format. SQL generates an error message if you attempt to exceed the precision specified.
- The USER keyword specifies the current active user name for a request.
- A warning is generated when a NULL value is eliminated from a SET function.
- The WITH CHECK OPTION clause on views returns a discrete error code from an integrity constraint failure.
- An exception is generated with non-null terminated C strings.

Table 7-5 shows the dialect settings for each environment.

**NAMES ARE names-char-set**
Specifies the character set used for the default, identifier, and literal character sets for the module. Also specifies the character string parameters that are not qualified by a character set or national character set. If you do not specify a character set, the default is DEC_MCS.

You must ensure that the character set specified in this clause matches the character set of all the databases attached to by any particular connection and must contain ASCII characters. See Table 2-3 for a list of the allowable character sets.

**LITERAL CHARACTER SET support-char-set**
Specifies the character set for literals that are not qualified by a character set or national character set. If you do not specify a character set in this clause or in the NAMES ARE clause, the default is DEC_MCS. This clause overrides the character set for unqualified literals specified in the NAMES ARE clause. See Section 2.1 for a list of the allowable character sets.

**NATIONAL CHARACTER SET support-char-set**
Specifies the character set for literals qualified by the national character set. See Section 2.1 for a list of the allowable character sets.

**DEFAULT CHARACTER SET support-char-set**
Specifies the character set for parameters that are not qualified by a character set. The default is DEC_MCS. This clause overrides the character set specified in the NAMES ARE clause. See Section 2.1 for a list of the allowable character sets.

**IDENTIFIER CHARACTER SET names-char-set**
Specifies the character set used for database object names such as table names and column names. This clause overrides the character set specified in the NAMES ARE clause. See Table 2-3 for a list of allowable character sets and option values.

The specified character set must contain ASCII characters.

**CATALOG catalog-name**
Specifies the default catalog for the module. **Catalogs** are groups of schemas within a multischema database. If you omit the catalog name when specifying an object in a multischema database, SQL uses the default catalog name RDB$CATALOG. Databases created without the multischema attribute do not have catalogs. You can use the SET CATALOG statement to change the current default catalog name in dynamic or interactive SQL.

**SCHEMA schema-name**
Specifies the default schema name for the module. The **default schema** is the schema to which SQL statements refer if those statements do not qualify table names and other schema names with an authorization identifier. If you do not specify a default schema name for a module, you must specify a default authorization identifier.

Using the SCHEMA clause, separate modules can each declare different schemas as default schemas. This can be convenient for an application that needs to refer to more than one schema. By putting SQL statements that refer to a schema in the appropriate module's procedures, you can minimize tedious qualification of schema element names in those statements.

When you specify SCHEMA schema-name AUTHORIZATION auth-id, you specify the schema name and the schema authorization identifier for the module. The schema authorization identifier is considered the owner and creator of the schema and everything in it.

**AUTHORIZATION auth-id**
Specifies the authorization identifier for the module. If you do not specify a schema clause, the authorization identifier specifies the default schema.

To comply with the ANSI/ISO 1989 standard, specify the AUTHORIZATION clause without the schema name. Specify both the AUTHORIZATION clause and the schema name to comply with the ANSI/ISO SQL standard.

When you attach to a multischema database, the authorization identifier for each schema is the user name of the user compiling the module. This authorization identifier defines the default alias and schema. You can use the SCHEMA clause and the DECLARE ALIAS statement to override the defaults.

If you attach to a single-schema database or specify that MULTISCHEMA IS OFF in your ATTACH or DECLARE ALIAS statements and you specify both an AUTHORIZATION clause and an ALIAS clause, the authorization identifier is ignored by SQL unless you use the RIGHTS RESTRICT clause. The RIGHTS RESTRICT clause causes SQL to use the authorization identifier specified in the module AUTHORIZATION clause for privilege checking.

If procedures in the SQL module always qualify table names with an authorization identifier, the AUTHORIZATION clause has no effect on SQL statements in the procedures.

When the FIPS flagger is enabled, the omission of an AUTHORIZATION clause is flagged as nonstandard ANSI syntax.

**ALIAS alias-name**
Specifies the module alias. If you do not specify a module alias, the default alias is the authorization identifier for the module.

When the FIPS flagger is enabled, the ALIAS clause (by itself or used with the AUTHORIZATION clause) is flagged as nonstandard syntax.

If the application needs to refer to only one database across multiple modules, it is good practice to use the same alias for the default database in all modules that will be linked to make up an executable image.

**CHARACTER LENGTH CHARACTERS**
**CHARACTER LENGTH OCTETS**
Specifies whether the length of character string parameters, columns, and domains are interpreted as characters or octets. The default is octets.

**DEFAULT DATE FORMAT SQL92**
**DEFAULT DATE FORMAT VMS**
Controls the default interpretation for columns with the DATE or CURRENT_TIMESTAMP data type. The DATE and CURRENT_TIMESTAMP data types can be either VMS or SQL92 format.

If you specify VMS, both data types are interpreted as VMS format. The VMS format DATE and CURRENT_TIMESTAMP contain YEAR TO SECOND fields, like a TIMESTAMP.

If you specify SQL92, both data types are interpreted as SQL92 format. The SQL92 format DATE contains only the YEAR TO DAY fields.

The default is VMS.

Use the DEFAULT DATE FORMAT clause, rather than the SQLOPTIONS = ANSI_DATE qualifier because the qualifier will be deprecated in a future release.

**QUOTING RULES**
Controls whether double quotation marks are interpreted as string literals or delimited identifiers. If you specify SQL92, SQL89, or MIA, SQL interprets double quotation marks as delimited identifiers. If you specify SQLV40, SQL interprets double quotation marks as literals. The default is SQLV40.

Use the QUOTING RULES clause, rather than the SQLOPTIONS = ANSI_QUOTING qualifier because the qualifier will be deprecated in a future release.

**KEYWORD RULES**
Controls whether or not identifiers can be keywords. If you specify SQL92, SQL89, or MIA, you cannot use keywords as identifiers, unless you enclose them in double quotation marks. If you specify SQLV40, you can use keywords as identifiers. The default is SQLV40.

Use the KEYWORD RULES clause, rather than the SQLOPTIONS = ANSI_IDENTIFIER qualifier because the qualifier will be deprecated in a future release.

**PARAMETER COLONS**
If you use the PARAMETER COLONS clause, all parameter names must begin with a colon ( : ). This is valid in context files for module language only. This rule applies to both declarations and references of module language procedure parameters. If you do not use this clause, no parameter name can begin with a colon.

The current default behavior is no colons are used. However, this default is deprecated syntax. In the future, required colons will be the default because it allows processing of ANSI/ISO SQL standard modules.

Use the PARAMETER COLONS clause, rather than the SQLOPTIONS = ANSI_PARAMETERS qualifier because the qualifier will be deprecated in a future release.

**RIGHTS INVOKER**
**RIGHTS RESTRICT**
Specifies whether or not a module must be executed by a user whose authorization identifier matches the module authorization identifier.

If you specify RESTRICT, SQL bases privilege checking on the default authorization identifier. The default authorization identifier is the authorization identifier of the user who compiles a module, unless you specify a different authorization identifier using an AUTHORIZATION clause in the module. The RESTRICT option causes SQL to compare the user name of the person who executes a module with the default authorization identifier and prevents any user other than one with the correct authorization identifier from invoking that module. All applications that use multischema restrict the invoker by default.

If you specify INVOKER, SQL bases the privilege on the authorization identifier of the user running the module. The default is INVOKER.

Use the RIGHTS clause, rather than the SQLOPTIONS = ANSI_AUTHORIZATION qualifier because the qualifier will be deprecated in a future release.

**VIEW UPDATE RULES**
Specifies whether or not the SQL module processor applies the ANSI/ISO SQL standard for updatable views to all views created during compilation.

If you specify SQL92, SQL89, or MIA, the SQL module processor applies that ANSI/ISO SQL standard for updatable views to all views created during compilation. Views that do not comply with the specified ANSI/ISO SQL standard for updatable views cannot be updated.

The specified ANSI/ISO standard for updatable views requires the following conditions to be met in the SELECT statement:

- The DISTINCT keyword is not specified.
- Only column names can appear in the select list. Each column name can appear only once. Functions and expressions such as max(column_name) or column_name +1 cannot appear in the select list.

- The FROM clause refers to only one table. This table must be either a base table or a derived table that can be updated.
- The WHERE clause does not contain a subquery.
- The GROUP BY clause is not specified.
- The HAVING clause is not specified.

If you specify SQLV40, SQL does not apply the ANSI/ISO standard for updatable views. Instead, SQL considers views that meet the following conditions to be updatable:

- The DISTINCT keyword is not specified.
- The FROM clause refers to only one table. This table must be either a base table or a derived table that can be updated.
- The WHERE clause does not contain a subquery.
- The GROUP BY clause is not specified.
- The HAVING clause is not specified.

## Example

Example 1: Declaring a module specifying character strings of different character sets

Assuming that the character sets for the database match the character sets specified in the program, the following example shows a simple SQL precompiled C program that retrieves one row from the COLOURS table.

```
/* This SQL precompiled program does some simple tests of character length
*  and character sets.
*/
#include stdio
#include descrip

main()
{

/* Specify CHARACTER LENGTH CHARACTERS in the DECLARE MODULE statement.
*  In addition, specify the NAMES, NATIONAL, and DEFAULT character sets.
*/
EXEC SQL DECLARE MODULE CCC_COLOURS
         NAMES ARE DEC_KANJI
         NATIONAL CHARACTER SET KANJI
         SCHEMA RDB$SCHEMA
         AUTHORIZATION SQL_SAMPLE
         CHARACTER LENGTH CHARACTERS
         DEFAULT CHARACTER SET DEC_KANJI
         ALIAS RDB$DBHANDLE;
/* If you do not specify character sets in the DECLARE ALIAS statement, SQL
*  uses the character sets of the compile-time database.
*/
EXEC SQL DECLARE ALIAS FILENAME MIA_CHAR_SET;

int      SQLCODE;

/* Because the default character set is DEC_KANJI, you do not need to qualify
*  the variable dec_kanji_p with the character set, but you must declare
*  char in lowercase.
*/
char  dec_kanji_p[31];

/* When you declare a parameter with lowercase char, SQL considers the
*  character set unspecified and allocates single-octet characters.
*/
char  english_p[31];

/* When you specify the character set, SQL allocates single- or multi-octet
*  characters, depending upon the character set.
*/
char  CHARACTER SET DEC_MCS  french_p[31];
char  CHARACTER SET KANJI    japanese_p[31];
   .
   .
   .
/* Select one row from the COLOURS table. */
EXEC SQL SELECT ENGLISH, FRENCH, JAPANESE, ROMAJI,
                KATAKANA, HINDI, GREEK, ARABIC, RUSSIAN
         INTO :english_p, :french_p, :japanese_p, :dec_kanji_p,
              :katakana_p, :devanagari_p, :isolatingreek_p,
              :isolatinarabic_p, :isolatincyrillic_p
         FROM COLOURS LIMIT TO 1 ROW;

     if (SQLCODE != 0)
         SQL$SIGNAL();

printf ("\nENGLISH: %s", english_p);
printf ("\nFRENCH: %s", french_p);
printf ("\nJAPANESE: %s", japanese_p);
printf ("\nROMAJI: %s", dec_kanji_p);
printf ("\nKATAKANA: %s", katakana_p);
printf ("\nHINDI: %s", devanagari_p);
printf ("\nGREEK: %s", isolatingreek_p);
printf ("\nARABIC: %s", isolatinarabic_p);
printf ("\nRUSSIAN: %s", isolatincyrillic_p);

EXEC SQL   ROLLBACK;
}
```

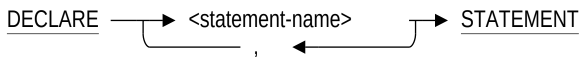
## DECLARE STATEMENT Statement

Documents a statement name later used in a PREPARE statement in dynamic SQL. SQL does not require DECLARE STATEMENT statements and does not generate any code when it precompiles them. They are entirely optional.

### Environment

You can issue the DECLARE STATEMENT statement only in host language programs to be precompiled.

### Format



### Arguments

**statement-name STATEMENT**
Specifies the name of a statement later referred to in one of the following embedded dynamic statements:

- PREPARE
- DECLARE CURSOR
- DESCRIBE

### Example

OpenVMS VAX OpenVMS Alpha

Example 1: Declaring a statement name in a PL/I program

This example shows a program line that declares a statement name DYNAMIC_STATEMENT. Later lines in the example show how DECLARE CURSOR, PREPARE, and DESCRIBE statements refer to it. Because you do not have to declare a statement explicitly, the DECLARE STATEMENT statement is always optional.

```
EXEC SQL DECLARE DYNAMIC_STATEMENT STATEMENT;
/* Declare the SQL Communications Area. */
EXEC SQL INCLUDE SQLCA;
/* Declare the SQL Descriptor Area. */
EXEC SQL INCLUDE SQLDA;

/* The program declares the host language variable
   STATEMENT_STRING and stores in it the
   character string containing a SELECT
   statement to be executed dynamically. */
                .
                .
                .
EXEC SQL DECLARE CURSOR1 CURSOR FOR DYNAMIC_STATEMENT;
EXEC SQL PREPARE OBJECT_STATEMENT FROM STATEMENT_STRING;
EXEC SQL DESCRIBE OBJECT_STATEMENT INTO SQLDA;

/* The program sets up pointers in the
   SQLDATA field of the SQLDA to the data
   area (host language variables or dynamic
   memory, for example) to receive the data
   from the cursor. */
                .
                .
                .
EXEC SQL OPEN CURSOR1;

DO WHILE (SQLCODE = 0);
        EXEC SQL FETCH CURSOR1 USING DESCRIPTOR SQLDA;

/* The program prints or otherwise
processes rows of the result tables. */
                .
                .
                .
END;

EXEC SQL CLOSE CURSOR1;
```
♦

## DECLARE TABLE Statement

Explicitly declares a table or view definition in a program. For tables named in a DECLARE TABLE statement, SQL does not check the schema to compare the definition with the explicit declaration.

An explicit table declaration is useful to:

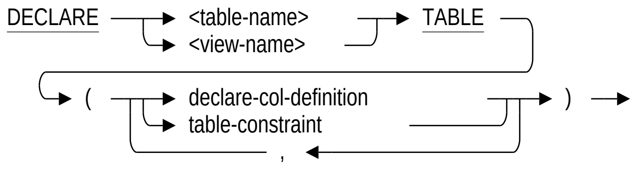
- Document the definition in the source code of the program
- Allow references to tables that do not exist when SQL precompiles the program, including:
  - Tables created in other modules of the program
  - Tables created dynamically
- Improve precompiler performance because SQL does not need to attach to the schema to retrieve the table definition
- Make it easier to check that the declaration correctly corresponds to a host structure the program uses to hold values from or for the table
- Declare only a subset of columns contained in the schema definition of the table if the program needs to use only some of the columns

### Environment

You can use the DECLARE TABLE statement:

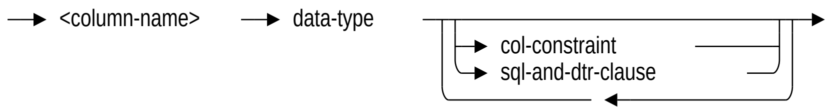
- Embedded in host language programs to be precompiled
- In a context file
- As part of the DECLARE section in an SQL module

### Format

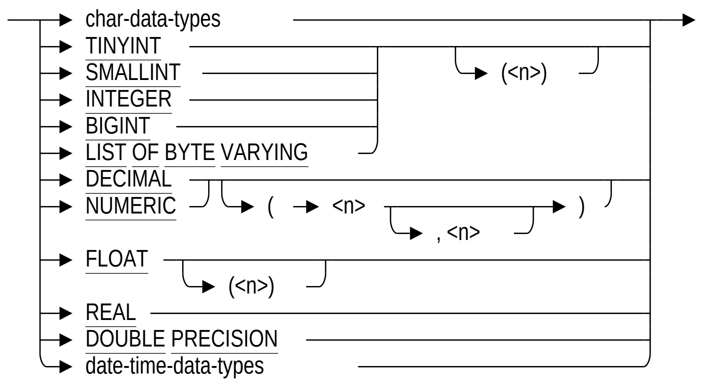


## DECLARE TABLE Statement

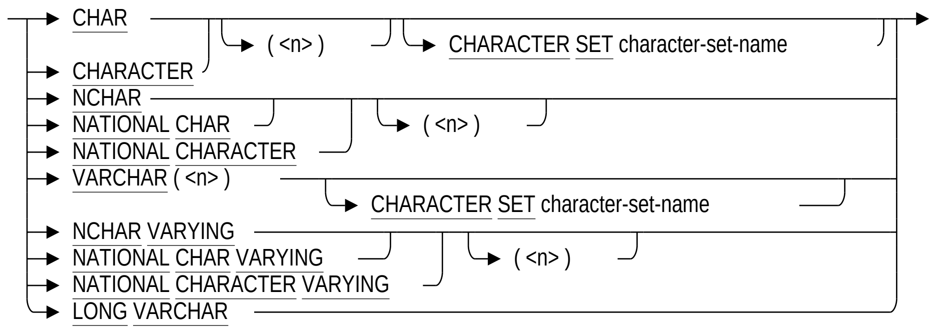
declare-col-definition =



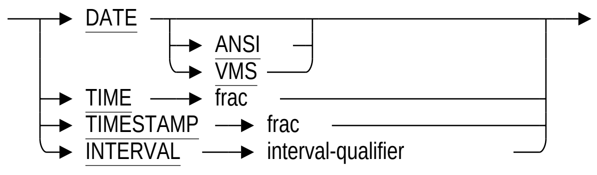
data-type =



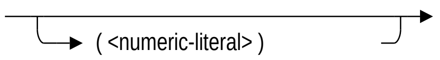
char-data-types =



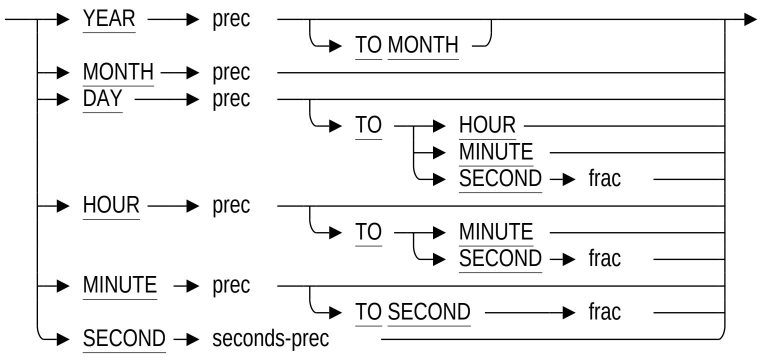
date-time-data-types =



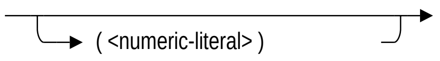
frac =



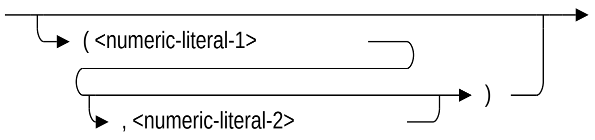
interval-qualifier =



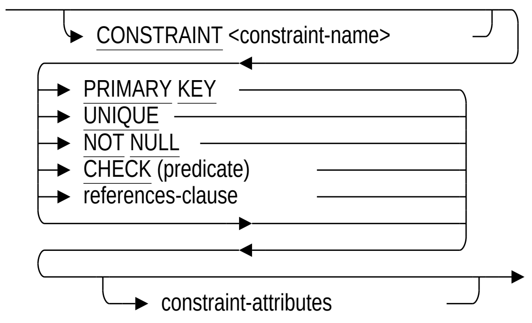
prec =



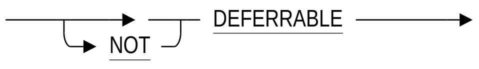
seconds-prec =



col-constraint=



constraint-attributes =



sql-and-dtr-clause =

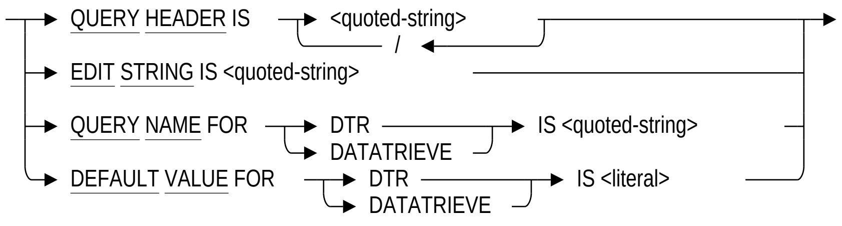


table-constraint =

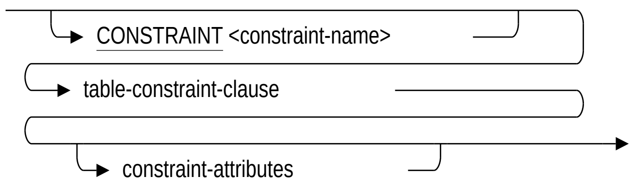
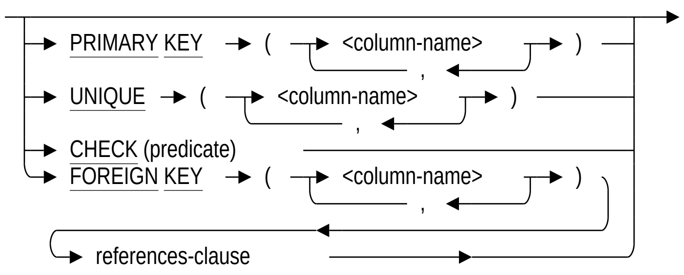


table-constraint-clause =



## Arguments

**table-name**
**view-name**
The name of the table or view definition you want to declare.

**declare-col-definition**
The definition for a column in the table. The column definition must correspond to the table definition in the schema.

See the CREATE TABLE Statement for more information about column definitions.

However, you cannot refer to domain names in a DECLARE TABLE statement. For tables whose definitions refer to domain names, you must substitute the data type and size of the domain for the domain name.

**column-name**
The name of the column you want to define.

**data-type**
The data type of the column you want to define. See Section 2.3 for more information on data types.

**character-set-name**
A valid character set name. See Section 2.1 for more information on character sets.

**date-time-data-types**
Data types for dates, times, and intervals. See Section 2.3.5 for more information on date-time data types.

**frac**
**interval-qualifier**
**prec**
**seconds-prec**
Precision specifications for date-time data types. See Section 2.3.5 for more information.

**col-constraint**
A column constraint. See the CREATE TABLE Statement for more information about column constraints.

**sql-and-dtr-clause**
Optional SQL and DATATRIEVE formatting clause. See Section 2.5 for more information about formatting clauses.

**table-constraint**
A constraint definition that applies to the whole table. See the CREATE TABLE Statement for more information about specifying table constraints.

## Usage Notes

SQL uses the declaration in the DECLARE TABLE statement when it precompiles embedded SQL statements or processes the module procedures that refer to the table. Therefore, the columns in the declaration should match the columns in the schema definition. However, the table or view definition to which the declaration in the DECLARE TABLE statement corresponds does not have to exist before a program can issue a DECLARE TABLE statement. The program can create the table after it declares it.

## Examples

Example 1: Declaring the table EMPLOYEES in a COBOL program

```
EXEC SQL
DECLARE EMPLOYEES TABLE
    (EMPLOYEE_ID                      CHAR (5)
       CONSTRAINT     EMP_EMPLOYEE_ID_NOT_NULL
       NOT NULL,
     LAST_NAME                        CHAR (14),
     FIRST_NAME                       CHAR (10),
     MIDDLE_INITIAL                   CHAR (1),
     ADDRESS_DATA_1                   CHAR (25),
     ADDRESS_DATA_2                   CHAR (25),
     CITY                             CHAR (20),
     STATE                            CHAR (2),
     POSTAL_CODE                      CHAR (5),
     SEX                              CHAR (1),
       CONSTRAINT     EMP_SEX_VALUES
       CHECK          (
                      SEX IN ('M', 'F') OR SEX IS NULL
                      ),
     BIRTHDAY                         DATE ,
     STATUS_CODE                      CHAR (1)
       CONSTRAINT     EMP_STATUS_CODE_VALUES
       CHECK          (
                      STATUS_CODE IN ('0', '1', '2')
                      OR STATUS_CODE IS NULL
                      )
     )
END_EXEC
```

## DECLARE TRANSACTION Statement

Specifies the characteristics for a default transaction. A **transaction** is a group of statements whose changes can be made permanent or undone only as a unit.

A transaction ends with a COMMIT or ROLLBACK statement. If you end the transaction with the COMMIT statement, all changes made to the database by the statements are made permanent. If you end the transaction with the ROLLBACK statement, the statements do not take effect.

The characteristics specified in a DECLARE TRANSACTION statement affect all transactions (except those started by the SET TRANSACTION statement) until you issue another DECLARE TRANSACTION statement. The characteristics specified in a SET TRANSACTION statement affect only that transaction.

A DECLARE TRANSACTION statement does not start a transaction. The declarations made in a DECLARE TRANSACTION statement do not take effect until SQL starts a new transaction. SQL starts a new transaction with the first executable data manipulation or data definition statement following a DECLARE TRANSACTION, COMMIT, or ROLLBACK statement. In the latter case (following a COMMIT or ROLLBACK statement), SQL applies the transaction characteristics you declared for the transaction that just ended to the next one you start.

In addition to the DECLARE TRANSACTION statement, you can specify the characteristics of a transaction in one of two ways:

- If you specify the SET TRANSACTION statement, the declarations in the statement take effect immediately and SQL starts a new transaction.
- You can retrieve and update data without declaring or setting a transaction explicitly. If you omit both the DECLARE TRANSACTION and SET TRANSACTION statements, SQL automatically starts a transaction (using the read/write option) with the first executable data manipulation or data definition statement following a COMMIT or ROLLBACK statement.

See the Usage Notes for examples of when you would want to use the DECLARE TRANSACTION statement instead of the SET TRANSACTION statement.

You can specify many options with the DECLARE TRANSACTION statement, including:

- A transaction mode (READ ONLY/READ WRITE/BATCH UPDATE)

- A lock specification clause (RESERVING options)
- A wait mode (WAIT/NOWAIT)
- An isolation level
- A constraint evaluation specification clause
- Multiple sets of all the preceding options for each database involved in the transaction (ON, AND ON)

## Environment

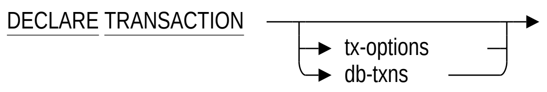
You can use the DECLARE TRANSACTION statement:

- In interactive SQL
- Embedded in host language programs to be precompiled
- In a context file
- As part of the DECLARE section in an SQL module
- In dynamic SQL as a statement to be dynamically executed

In host language programs, you can have only a single DECLARE TRANSACTION statement in each separately compiled source file. See the Usage Notes for more information.

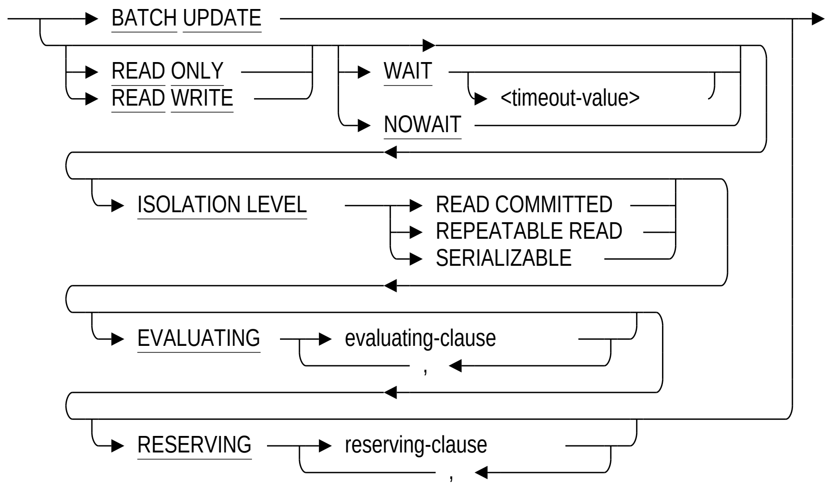
The DECLARE TRANSACTION statement is an extension to standard SQL syntax. If your program must adhere to standard SQL syntax, you can isolate a DECLARE TRANSACTION statement by putting it in a context file. For more information on context files, see the *Oracle Rdb7 Guide to SQL Programming*.

## Format

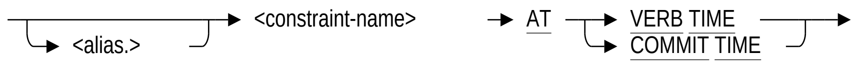


## DECLARE TRANSACTION Statement

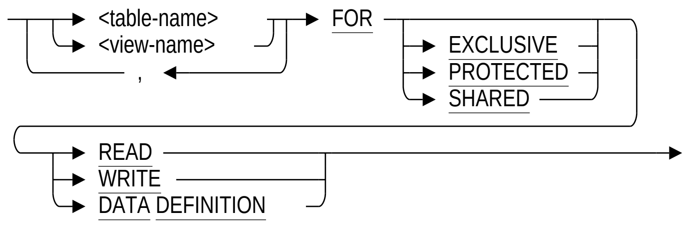
tx-options =



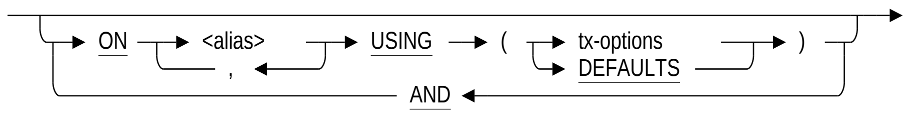
evaluating-clause =



reserving-clause =



db-txns =



## Arguments

The DECLARE TRANSACTION arguments are the same as the arguments for the SET TRANSACTION statement. See the SET TRANSACTION Statement for more information about the arguments for both statements.

## Defaults

The DECLARE TRANSACTION defaults are the same as the defaults for the SET TRANSACTION statement. See the SET TRANSACTION Statement for complete information.

In general, you should not rely on default transaction characteristics. Use explicit DECLARE TRANSACTION statements, specifying read/write, read-only, or batch-update options; a list of tables in the RESERVING clause; and a share mode and lock type for each table. The more specific you are in a DECLARE TRANSACTION statement, the more efficient your database operations will be.

When a transaction starts using characteristics specified in a DECLARE TRANSACTION statement, any transaction characteristics unspecified in the DECLARE TRANSACTION statement take the SQL defaults. This is true even if the characteristics unspecified in DECLARE TRANSACTION were specified in an earlier SET or DECLARE TRANSACTION statement.

## Usage Notes

The following notes are particular to DECLARE TRANSACTION. See the SET TRANSACTION Statement for usage notes that are common to both DECLARE TRANSACTION and SET TRANSACTION statements.

- The DECLARE TRANSACTION statement is not executable, and therefore, does not start a transaction. (The declarations in a DECLARE TRANSACTION statement take effect when SQL starts a new transaction; that is, with the first executable data manipulation or data definition statement following the DECLARE TRANSACTION, COMMIT, or ROLLBACK statement.)

  You can apply only one DECLARE TRANSACTION statement to a host language source file or to an SQL module. Use the SET TRANSACTION statement to change transaction characteristics in programs that were first specified using the DECLARE TRANSACTION statement.

The following are advantages offered by the DECLARE TRANSACTION statement:

– It can establish transaction defaults for an interactive SQL session, a module or single host language file in a program, or any statements executed dynamically from a module. You might, for example, specify DECLARE TRANSACTION READ ONLY in the SQLINI.SQL file you create to set up your interactive SQL environment.

  In interactive SQL, the characteristics specified by a DECLARE TRANSACTION statement are valid until you enter another DECLARE TRANSACTION statement. (A COMMIT or ROLLBACK statement followed by a SET TRANSACTION statement may start a transaction with different characteristics, but subsequent transactions started implicitly will have the characteristics specified in the last DECLARE TRANSACTION statement.)

  If you specify characteristics using a SET TRANSACTION statement, however, the characteristics apply only to that transaction. You must reenter the statement after every COMMIT or ROLLBACK statement to establish those characteristics again.

  The following sequence shows a DECLARE TRANSACTION statement followed by a SET TRANSACTION statement. Note that the SET TRANSACTION statement is followed by a ROLLBACK statement:

```
SQL> -- Declares default characteristics for transactions:
SQL> --
SQL> DECLARE TRANSACTION READ WRITE;
SQL> --
SQL> -- There is no transaction started; can start
SQL> -- transaction with characteristics different
SQL> -- from the declared characteristics using a
SQL> -- SET TRANSACTION statement:
SQL> --
SQL> SET TRANSACTION READ ONLY;
SQL> --
SQL> -- Roll back the transaction started by
SQL> -- the SET TRANSACTION statement:
SQL> --
SQL> ROLLBACK;
SQL> --
SQL> -- The default transaction characteristics are still those
SQL> -- specified in the DECLARE TRANSACTION statement, and
SQL> -- apply to the transaction started when this SELECT
SQL> -- statement executes:
SQL> --
SQL> SELECT * FROM EMPLOYEES;
```

  - You can include the DECLARE TRANSACTION statement in an SQL context file.

    The section in the *Oracle Rdb7 Guide to SQL Programming* about program transportability explains when you may need an SQL context file to support a program that includes SQL statements.

- In contrast to the DECLARE TRANSACTION statement, SET TRANSACTION is an executable statement that specifies and starts one transaction. You can include multiple SET TRANSACTION statements in a host language source file or in an SQL language module. The SET TRANSACTION statement has the following advantages:

  - It gives you explicit control over when transactions are started.
  - It provides flexibility for changing transaction characteristics in a program source file.

- In precompiled programs, you can have only a single DECLARE TRANSACTION statement in each separately compiled source file. It must precede any executable SQL statement and follow all DECLARE ALIAS statements. This restriction is not enforced for dynamically executed DECLARE TRANSACTION statements.

  You can include multiple DECLARE TRANSACTION statements in a program by linking multiple, separately compiled modules, each with an associated DECLARE TRANSACTION statement. However, the transaction characteristics that the statements specify will not necessarily apply to their modules.

  At execution time, when any module starts a transaction, the characteristics declared by that module apply to all modules until the transaction ends. In other words, the DECLARE TRANSACTION statement only specifies characteristics for implicit transactions started by that module; it does not ensure that those characteristics are current when execution begins. Depending on the execution path of your program, this may make it difficult to control the transaction characteristics that apply to a particular module. For instance, if a module does not have an explicit DECLARE TRANSACTION statement and that module starts a transaction, default transaction characteristics apply to all modules until the transaction ends.

  When it is important to have particular transaction characteristics apply to a given module, you must be careful to end transactions before program control branches to that module. The SET TRANSACTION statement is best suited to this situation.

- When you use the AND ON clause to start a transaction for more than one database, you should make sure that the DECLARE TRANSACTION statement includes an ON clause for every attached database. If you do not, you cannot use or refer to the databases omitted from the DECLARE TRANSACTION statement in any SQL statement, including SHOW and later DECLARE TRANSACTION statements.

  If you do refer to a database that is attached but not included in a multiple database DECLARE TRANSACTION statement, this message is generated:

  ```
  %RDB-F-REQ_WRONG_DB, database named in specified request is not
     a database named in specified transaction
  ```

- If you use the BATCH UPDATE clause with a DECLARE TRANSACTION statement, SQL returns an error at compile time. See the SET TRANSACTION Statement for more information about the BATCH UPDATE clause.
- If you use the DECLARE TRANSACTION statement in a stored module with either the RESERVING table clause or the EVALUATING constraint clause, SQL establishes dependencies on the tables or constraints that you specify. See the CREATE MODULE Statement for a list of statements that can or cannot cause stored procedure invalidation.

  See the *Oracle Rdb7 Guide to SQL Programming* for detailed information about stored procedure dependency types and how metadata changes can cause invalidation of stored procedures.

## Examples

Example 1: Illustrating DECLARE and SET TRANSACTION differences

In the following example, the first executable statement following the DECLARE TRANSACTION statement starts a transaction. In contrast, the subsequent SET TRANSACTION statement itself starts a transaction.

```
SQL> DECLARE TRANSACTION READ WRITE NOWAIT;
SQL> --
SQL> -- Notice the "no transaction is in progress" message:
SQL> --
SQL> SHOW TRANSACTION
Transaction information:
    Statement constraint evaluation is off
```

```
On the default alias
Transaction characteristics:
        Nowait
        Read Write
Transaction information returned by base system:
no transaction is in progress
  - session ID number is 80
SQL> --
SQL> -- The first executable statement following the
SQL> -- DECLARE TRANSACTION statement starts the transaction.
SQL> -- In this case, SELECT is the first executable statement.
SQL> --
SQL> SELECT LAST_NAME FROM CURRENT_SALARY;
 LAST_NAME
 Toliver
 Smith
 Dietrich
   .
   .
   .
SQL> --
SQL> -- Note the transaction inherits the read/write characteristics
SQL> -- specified in the DECLARE TRANSACTION statement:
SQL> --
SQL> SHOW TRANSACTION;

Transaction information:
    Statement constraint evaluation is off

On the default alias
Transaction characteristics:
        Nowait
        Read Write
Transaction information returned by base system:
a read-write transaction is in progress
  - updates have not been performed
  - transaction sequence number (TSN) is 416
  - snapshot space for TSNs less than 416 can be reclaimed
  - session ID number is 80
SQL> --
SQL> ROLLBACK;
SQL> --
SQL> -- Again, notice the "no transaction is in progress" message:
SQL> --
SQL> SHOW TRANSACTION;

Transaction information:
    Statement constraint evaluation is off
```

```
On the default alias
Transaction characteristics:
        Nowait
        Read Write
Transaction information returned by base system:
no transaction is in progress
  - transaction sequence number (TSN) 416 is reserved
  - snapshot space for TSNs less than 416 can be reclaimed
  - session ID number is 80
SQL> --
SQL> -- Unlike DECLARE TRANSACTION, the SET TRANSACTION statement
SQL> -- immediately starts a transaction:
SQL> --
SQL> SET TRANSACTION READ ONLY WAIT;
SQL> --
SQL> -- Note the transaction characteristics show the
SQL> -- read-only characteristics:
SQL> --
SQL> SHOW TRANSACTION;
Transaction information:
    Statement constraint evaluation is off

On the default alias
Transaction characteristics:
        Wait
        Read only
Transaction information returned by base system:
a snapshot transaction is in progress
  - all transaction sequence numbers (TSNs) less than 416 are visible
  - TSN 416 is invisible
  - all TSNs greater than or equal to 417 are invisible
  - session ID number is 80
```

Example 2: Using a DECLARE TRANSACTION statement in a context file

The following example shows a context file, test_declares.sql, that contains declarations for precompiling source file test.sco:

```
DECLARE ALIAS FOR FILENAME personnel;
DECLARE TRANSACTION READ WRITE
        RESERVING EMPLOYEES FOR PROTECTED WRITE,
                  JOB_HISTORY FOR PROTECTED WRITE,
                  DEPARTMENTS FOR SHARED READ,
                  JOBS FOR SHARED READ;
```

The section in the *Oracle Rdb7 Guide to SQL Programming* about program transportability explains when you may need an SQL context file to support a program that includes SQL statements.

Example 3: Explicitly setting the isolation level in a DECLARE TRANSACTION statement

In this example, you declare the default characteristics for a read/write transaction, which includes changing the default ISOLATION LEVEL SERIALIZABLE to ISOLATION LEVEL REPEATABLE READ.

```
SQL> DECLARE TRANSACTION READ WRITE ISOLATION LEVEL REPEATABLE READ;
```

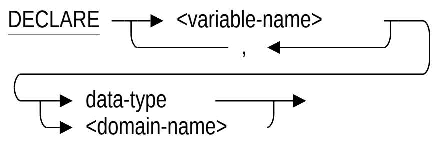
## DECLARE Variable Statement

Declares variables that you can use in interactive SQL for invoking stored procedures and for testing procedures in modules or embedded SQL programs. For information on declaring variables in compound statements, see the Compound Statement.

### Environment

You can issue the DECLARE variable statement only in interactive SQL.

### Format



### Arguments

**variable-name**
Specifies the local variable for interactive SQL.

**data-type**
Specifies the data type assigned to the variable. See Section 2.3 for more information on data types.

**domain-name**
Specifies the domain name assigned to the variable. See Section 2.2.11 for more information on domain names.

### Usage Notes

- Variables inside compound statements can be set to NULL. Interactive variables are more like host variables or parameters. You must use indicator variables to set interactive SQL variables to NULL.
- Variables exist until the end of the session or until the UNDECLARE Variable statement is executed. See the UNDECLARE Variable Statement for more information about deleting variable definitions.

- Use the SHOW VARIABLES statement to show the existing variable definitions.

## Example

Example 1: Declaring variables in interactive SQL

```
SQL> DECLARE :X INTEGER;
SQL> DECLARE :Y CHAR(10);
SQL>
SQL>  BEGIN
cont>    SET :X = 100;
cont>    SET :Y = 'Active';
cont> END;
SQL> PRINT :X, :Y ;
          X   Y
        100   Active
SQL> SHOW VARIABLES;
X                               INTEGER
Y                               CHAR(10)
```

# Index