Oracle Rdb™

SQL Reference Manual
Volume 2

Release 7.3.3.0 for HP OpenVMS Industry Standard 64 for Integrity Servers and OpenVMS Alpha operating systems

November 2018

ORACLE®

SQL Reference Manual, Volume 2

Oracle Rdb Release 7.3.3.0 for HP OpenVMS Industry Standard 64 for Integrity Servers and OpenVMS Alpha operating systems

Copyright © 1987, 2018 Oracle Corporation. All rights reserved.

Primary Author: Rdb Engineering and Documentation group

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle, Java, Oracle Rdb, Hot Standby, LogMiner for Rdb, Oracle SQL/Services, Oracle CODASYL DBMS, Oracle RMU, Oracle CDD/Repository, Oracle Trace, and Rdb7 are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Send Us Your Comments vii

Preface ix

6 SQL Statements

ACCEPT Statement 6–2
ALTER Statements 6–7
ALTER CONSTRAINT Statement 6–8
ALTER DATABASE Statement 6–14
ALTER DOMAIN Statement 6–92
ALTER FUNCTION Statement 6–110
ALTER INDEX Statement 6–118
ALTER MODULE Statement 6–142
ALTER OUTLINE Statement 6–152
ALTER PROCEDURE Statement 6–158
ALTER PROFILE Statement 6–164
ALTER ROLE Statement 6–169
ALTER SEQUENCE Statement 6–172
ALTER STORAGE MAP Statement 6–181
ALTER SYNONYM Statement 6–197
ALTER TABLE Statement 6–199
ALTER TRIGGER Statement 6–245
ALTER USER Statement 6–249
ALTER VIEW Statement 6–252
ATTACH Statement 6–258
BEGIN DECLARE Statement 6–269
CALL Statement for Simple Statements 6–272
CALL Statement for Compound Statements 6–275

CASE (Searched) Control Statement 6–279
CASE (Simple) Control Statement 6–281
CLOSE Statement 6–284
COMMENT ON Statement 6–286
COMMIT Statement 6–294
Compound Statement 6–301
CONNECT Statement 6–318
CREATE Statements 6–331
CREATE CACHE Clause 6–332
CREATE CATALOG Statement 6–344
CREATE COLLATING SEQUENCE Statement 6–348
CREATE DATABASE Statement 6–353
CREATE DOMAIN Statement 6–419
CREATE FUNCTION Statement 6–434
CREATE INDEX Statement 6–435
CREATE MODULE Statement 6–464
CREATE OUTLINE Statement 6–490
CREATE PROCEDURE Statement 6–517
CREATE PROFILE Statement 6–518
CREATE ROLE Statement 6–524
CREATE ROUTINE Statement 6–529
CREATE SCHEMA Statement 6–550

Index

Tables

6–1 Updating Data Definitions While Users Are Attached to the Database 6–69
6–2 Updating Database-Wide Parameters While Users Are Attached to the Database 6–72
6–3 Constraint Attributes Syntax Permutations and Equivalents 6–210
6–4 ALTER and DROP Statements Causing or Not Causing Stored Routine Invalidation 6–473

Send Us Your Comments

Oracle Rdb for OpenVMS
Oracle SQL Reference Manual, Release 7.3.3.0

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.

- 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?

If you find any errors or have any other suggestions for improvement, please indicate the document title, chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail:InfoRdb_US@oracle.com
- FAX — 603-897-3825 Attn: Oracle Rdb
- Postal service:
  Oracle Corporation
  Oracle Rdb Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual describes the syntax and semantics of 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 Rdb Installation and Configuration Guide*.

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

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Structure

This manual is divided into five 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 and an index. Volume 4 contains Chapter 8 and an index. Volume 5 contains 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, 3, 4, and 5 of the *Oracle Rdb SQL Reference Manual*:

Chapter 1	Introduces SQL (structured query language) and briefly describes SQL functions. This chapter also describes conformance to the SQL database 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 Chapter 7 Chapter 8	Describe 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 standards to which Oracle Rdb conforms.
Appendix C	Describes the SQL Communications Area, the message vector, and 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 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 compatibility with Oracle Database SQL. This appendix also describes the SQL syntax for performing an outer join between tables.
Appendix H	Describes the Oracle Rdb system tables.
Appendix I	Describes information tables that can be used with Oracle Rdb.
Index	Index for each volume.

Related Manuals

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

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

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.

The following conventions are also used in this manual:

. . .	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.

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, release 7.3 of Oracle Rdb software is often referred to as Version 7.3 or V7.3.
- 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 I64 refers to HP OpenVMS Industry Standard 64 for Integrity Servers.
- OpenVMS means the OpenVMS I64 and OpenVMS Alpha operating systems.

6
SQL Statements

This chapter describes the syntax and semantics of 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.

Chapter 7 in Volume 3 describes the statements from CREATE SEQUENCE to GRANT.

Chapter 8 in Volume 4 describes the statements from HELP to WHILE.

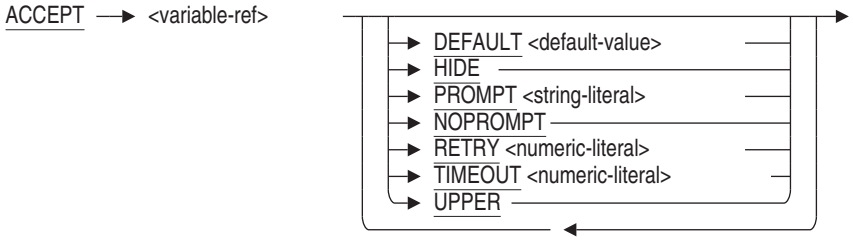
ACCEPT Statement

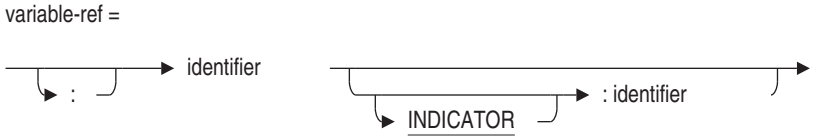
Prompts the user for additional information. This information is stored in an interactive SQL variable, which can subsequently be used by DML and some SET statements.

Environment

You can use the ACCEPT statement in interactive SQL.

Format





Arguments

DEFAULT default-value
Provides a default value to be used if the user presses the Return key. The default value must be a correctly formatted character string that can be converted to the data type of the variable.

HIDE
Disables echo of the input text. The default is to echo all input characters.

PROMPT string-literal
Provides a prompt string that is displayed before accepting input.

NOPROMPT
Disables prompting with a string.

RETRY numeric-literal
Reprompts the user for data input if an error occurs converting the input to the target data type.

TIMEOUT numeric-literal
If the user does not respond within this many seconds, then an error is returned. Negative or zero values of the numeric-literal are ignored. The default is to wait indefinitely.

UPPER
All lowercase characters are converted to uppercase before assignment to the variable. The default is to leave lowercase characters unchanged.

variable-ref
An interactive SQL variable defined using the DECLARE Variable statement.

Usage Notes

- The variable must be declared using the DECLARE syntax in interactive SQL. ACCEPT does not create this variable automatically. The leading colon (:) required for most variables references is optional in the ACCEPT command.

  The variable reference may include an indicator variable that can be used to detect NULL value assignments.

- When PROMPT is specified, the character string literal (up to a maximum of 80 octets) is used to prompt the user for input.

  When NOPROMPT is specified, then the user is not given a prompt string.

  If neither PROMPT nor NOPROMPT is specified, SQL will create a prompt containing the name of the variable. For instance:

```
SQL> ACCEPT :NAME;
Enter a value for NAME: Jeff
```

  The prompt string does not appear in the output created by the SET OUTPUT statement. Use the PRINT command to verify the input in such cases.

- This statement is based upon the ACCEPT statement of Oracle SQL*Plus. The following SQL*Plus clauses are not currently available in Oracle Rdb SQL: NUMBER, CHAR, DATE, and FORMAT.

- If the user enters no data, but presses the Return key, then the DEFAULT value is used in place of the response. If there was no DEFAULT specified, then a zero length string value is used, which may result in a valid value for numeric variables (zero) and string variables (spaces or the empty string), but will result in errors for DATE, TIME, TIMESTAMP, and INTERVAL variables.
- If the user enters EXIT (Ctrl/Z), then either an error is raised, or if an indicator variable is provided, then the value of the variable will be set to -1 (indicating NULL). The value of the variable will be undefined.

```
SQL> ACCEPT :id INDICATOR :id_ind PROMPT 'Id? ';
Id? <exit>

SQL> PRINT :id, :id_ind;
          ID        ID_IND
           0            -1
```

- If a timeout occurs and an indicator variable is provided, then the command succeeds and the indicator is set to -1 (indicating NULL). The value of the variable will be undefined.

```
SQL> declare :row_out, :row_out_ind integer;
SQL> accept :row_out timeout 5;
Enter value for ROW_OUT:
%SQL-F-UNDEFVAR, Variable ROW_OUT is not defined
-SYSTEM-F-TIMEOUT, device timeout
SQL> accept :row_out indicator :row_out_ind timeout 5;
Enter value for ROW_OUT:
SQL> print :row_out indicator :row_out_ind;
     ROW_OUT
        NULL
```

- If incompatible or incorrectly formatted data is entered, then an error will be issued and the contents of the variable will be unchanged.

```
SQL> DECLARE :age INTEGER;
SQL> ACCEPT :age;
Enter value for AGE: thirty
%RDB-E-ARITH_EXCEPT, truncation of a numeric value at runtime
-COSI-F-INPCONERR, input conversion error
```

- Values for DATE VMS, DATE ANSI, TIME, TIMESTAMP, and INTERVAL data types must include all required punctuation.

```
SQL> DECLARE :end_time TIME(2);
SQL> ACCEPT :end_time;
Enter value for END_TIME: 10:30
%SQL-F-DATCONERR, Data conversion error for string '10:30'
-COSI-F-IVTIME, invalid date or time
```

In this example, the required second and fractional seconds fields of the value were omitted.

- ACCEPT reads from SYS$COMMAND, which often defaults to the same input source as SYS$INPUT. However, it is possible to have SQL read from a separate source. In this example, SQL reads the SQL statements from SUMMARY_REPORT.SQL and accepts the answers from the file ANSWERS.DAT.

  ```
  $ define/user sys$command answers.dat
  $ sql$ @summary_report
  ```

  When the input source is not an interactive device, the PROMPT clause is ignored and the prompt does not appear in the output.

- Up to 20 previous inputs are available for command recall.
- If RETRY is omitted, then it defaults to zero (0), meaning that no retry is attempted upon a data conversion error.

Examples

Example 1: Prompting Based on the PROMPT and NOPROMPT Clauses

```
SQL> DECLARE :x INTEGER;
SQL> DECLARE :y INTEGER;
SQL>
SQL> ACCEPT :x indicator :y PROMPT 'what value? ';
what value? 10
SQL> PRINT :x, :y;
           X             Y
          10             0
SQL>
SQL> ACCEPT :x INDICATOR :y NOPROMPT;
11
SQL> PRINT :x, :y;
           X             Y
          11             0
SQL>
SQL> ACCEPT :x;
Enter value for X: 12
SQL> PRINT :x;
           X
          12
SQL>
```

Example 2: Using ACCEPT to Prompt for SET FLAGS String

This sequence would be included in a script.

```
SQL> DECLARE :debug_flags CHAR(20);
SQL> ACCEPT :debug_flags;
Enter value for DEBUG_FLAGS: trace
SQL> PRINT :debug_flags;
 DEBUG_FLAGS
 trace
SQL> SET FLAGS :debug_flags;
SQL> SHOW FLAGS

Alias RDB$DBHANDLE:
Flags currently set for Oracle Rdb:
   PREFIX,TRACE,MAX_RECURSION(100)
```

Example 3: Using RETRY to prompt the user for data again after erroneous input.

```
SQL> declare :v integer = 0;
SQL> accept :v retry 5;
Enter value for V: xxxx
%RDB-E-ARITH_EXCEPT, truncation of a numeric value at runtime
-COSI-F-INPCONERR, input conversion error
Enter value for V: 42
SQL> print :v;
           V
          42
SQL>
```

ALTER Statements

Modifies the database object.

Usage Notes

The following notes apply to all ALTER statements, except ALTER DATABASE.

- You cannot execute an ALTER statement when any of the LIST, DEFAULT or RDB$SYSTEM storage areas are set to read-only. You must first set these storage areas to read/write. Note that in some databases RDB$SYSTEM will also be the default and list storage area.
- The ALTER 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
```

- You must execute the ALTER 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.

ALTER CONSTRAINT Statement

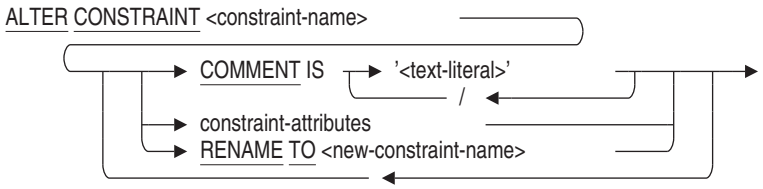
Alters attributes of a constraint created by the CREATE TABLE or ALTER TABLE statements.

Environment

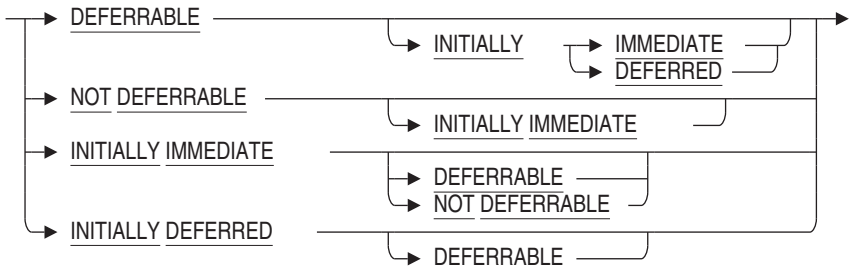
You can use the ALTER CONSTRAINT statement:

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

Format



constraint-attributes =



Arguments

COMMENT IS 'string'
Adds a comment about the constraint. SQL displays the text of the comment when it executes a SHOW CONSTRAINTS statement. Enclose the comment in

single quotation marks (′) and separate multiple lines in a comment with a slash mark (/).

constraint-attributes
See the ALTER TABLE Statement.

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

RENAME TO
Changes the name of the constraint being altered. See the RENAME Statement for further discussion. If the new name is the name of a synonym then an error will be raised.

The RENAME TO clause requires synonyms be enabled for this database. Refer to the ALTER DATABASE Statement SYNONYMS ARE ENABLED clause. Note that these synonyms may be deleted if they are no longer used by database definitions or applications.

Usage Notes

- The constraint name can be prefixed with an alias name. For example,

```
SQL> alter constraint db1.ALL_UNIQUE
cont>     deferrable initially deferred;
```

- The ALTER CONSTRAINT statement causes the RDB$LAST_ALTERED column of the RDB$CONSTRAINTS table for the named constraint to be updated with the transactions timestamp.
- The attributes that were defined or defaulted for a constraint can be changed without needing a drop and redefinition of the constraint. Existing compiled queries are not affected by this change. That is, they will use the constraint attributes that were current when the query was compiled.
- The ALTER CONSTRAINT Statement cannot be used to modify attributes of DOMAIN constraints. Use the ALTER DOMAIN Statement for such changes.
- Note that the constraint attributes replace the existing definition. This includes applying defaults within the ALTER CONSTRAINT statement.

Examples

Example 1

This example shows how ALTER CONSTRAINT can be used to change the constraint attributes and add a comment to a constraint.

```
SQL> set dialect 'sql99';
SQL> attach 'file db$:mf_personnel';
SQL>
SQL> create table PERSON
cont>     (last_name  char(20)
cont>         constraint MUST_HAVE_LAST_NAME
cont>             not null
cont>             deferrable,
cont>      first_name char(20),
cont>      birthday   date
cont>         constraint MUST_BE_IN_PAST
cont>             check (birthday < current_date)
cont>             not deferrable,
cont>      constraint ALL_UNIQUE
cont>         unique (last_name, first_name, birthday)
cont>         deferrable initially immediate
cont>     );
SQL>
SQL> show table (constraint) PERSON
Information for table PERSON

Table constraints for PERSON:
ALL_UNIQUE
 Unique constraint
     Null values are considered distinct
 Table constraint for PERSON
 Evaluated on each VERB
 Source:
 UNIQUE (last_name, first_name, birthday)

MUST_BE_IN_PAST
 Check constraint
 Column constraint for PERSON.BIRTHDAY
 Evaluated on UPDATE, NOT DEFERRABLE
 Source:
 CHECK (birthday < current_date)

MUST_HAVE_LAST_NAME
 Not Null constraint
 Column constraint for PERSON.LAST_NAME
 Evaluated on COMMIT
 Source:
 PERSON.LAST_NAME NOT null
```

```
Constraints referencing table PERSON:
No constraints found

SQL>
SQL> alter constraint ALL_UNIQUE
cont>     deferrable initially deferred;
SQL>
SQL> alter constraint MUST_HAVE_LAST_NAME
cont>     comment is 'We must assume all persons have a name'
cont>     not deferrable;
SQL>
SQL> alter constraint MUST_BE_IN_PAST
cont>     deferrable initially immediate;
SQL>
SQL> show table (constraint) PERSON
Information for table PERSON

Table constraints for PERSON:
ALL_UNIQUE
 Unique constraint
     Null values are considered distinct
 Table constraint for PERSON
 Evaluated on COMMIT
 Source:
 UNIQUE (last_name, first_name, birthday)

MUST_BE_IN_PAST
 Check constraint
 Column constraint for PERSON.BIRTHDAY
 Evaluated on each VERB
 Source:
 CHECK (birthday < current_date)

MUST_HAVE_LAST_NAME
 Not Null constraint
 Column constraint for PERSON.LAST_NAME
 Evaluated on UPDATE, NOT DEFERRABLE
 Comment: We must assume all persons have a name
 Source:
 PERSON.LAST_NAME NOT null

Constraints referencing table PERSON:
No constraints found

SQL>
SQL> commit;
```

Example 2

Adding a comment to a constraint.

```
SQL> alter constraint JOB_HISTORY_FOREIGN1
cont>  comment is 'Only reference valid employees';
SQL> show table (constraint) employees
Information for table EMPLOYEES
.
.
.
Constraints referencing table EMPLOYEES:
.
.
.
JOB_HISTORY_FOREIGN1
 Foreign Key constraint
 Column constraint for JOB_HISTORY.EMPLOYEE_ID
 Evaluated on COMMIT
 Comment:       Only reference valid employees
 Source:
        JOB_HISTORY.EMPLOYEE_ID REFERENCES EMPLOYEES (EMPLOYEE_ID)
.
.
.
```

Example 3

Changing a CONSTRAINT from DEFERRABLE to NOT DEFERRABLE

```
SQL> show table (constraint) employees
Information for table EMPLOYEES

Table constraints for EMPLOYEES:
.
.
.
EMP_STATUS_CODE_VALUES
 Check constraint
 Table constraint for EMPLOYEES
 Evaluated on COMMIT
 Source:
        CHECK          (
                         STATUS_CODE IN ('0', '1', '2', 'N')
                         )
.
.
.
SQL> alter constraint EMP_STATUS_CODE_VALUES not deferrable;
SQL> show table (constraint) employees
Information for table EMPLOYEES
```

```
Table constraints for EMPLOYEES:
.
.
.
EMP_STATUS_CODE_VALUES
 Check constraint
 Table constraint for EMPLOYEES
 Evaluated on UPDATE, NOT DEFERRABLE
 Source:
        CHECK          (
                         STATUS_CODE IN ('0', '1', '2', 'N')
                         )
.
.
.
```

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.

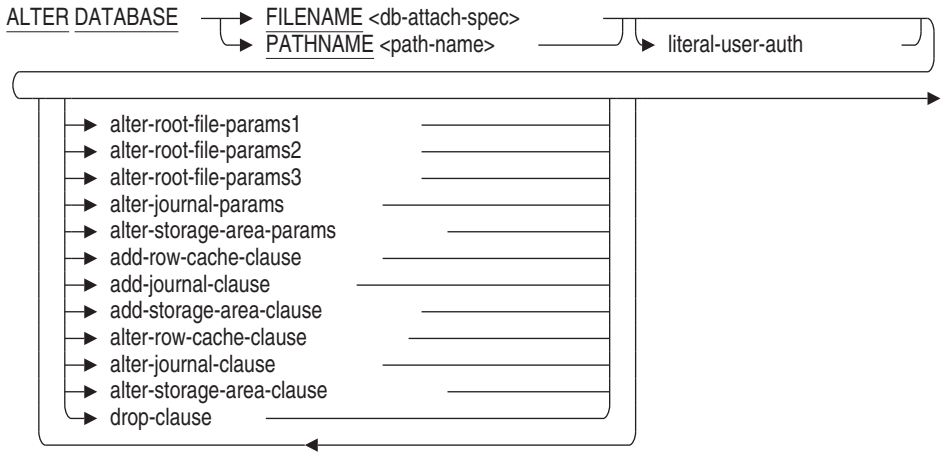
  The ALTER DATABASE statement lets you override certain characteristics specified in the database root file parameters of the CREATE DATABASE statement, such as whether or not a snapshot file is disabled. In addition, ALTER DATABASE lets you control other characteristics that you cannot specify in the 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:

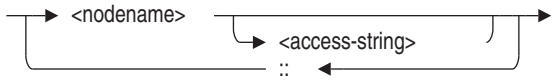
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

Format

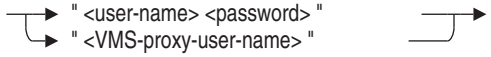


db-attach-spec =

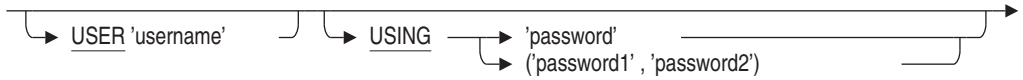
node-spec =



access-string =

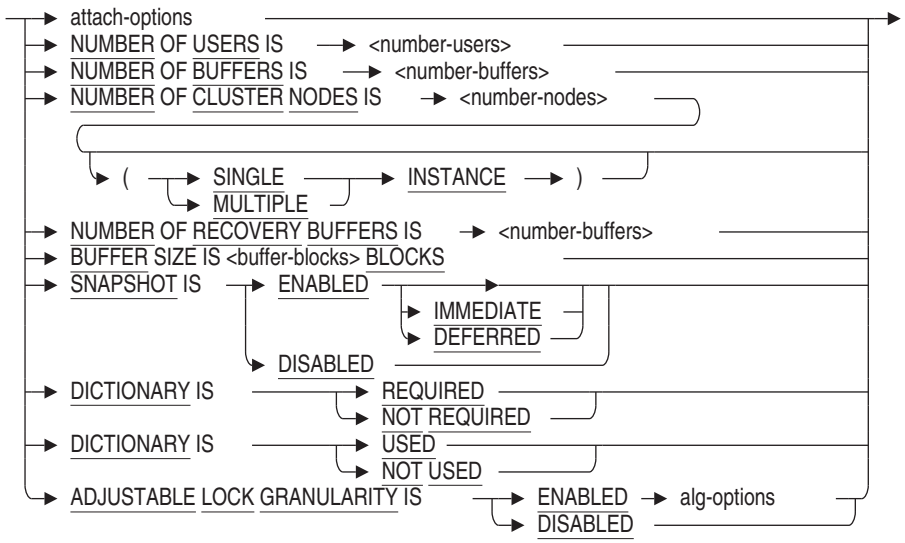


literal-user-auth =

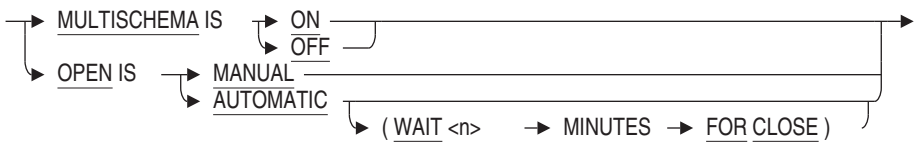


ALTER DATABASE Statement

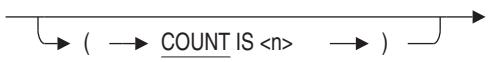
alter-root-file-params1 =



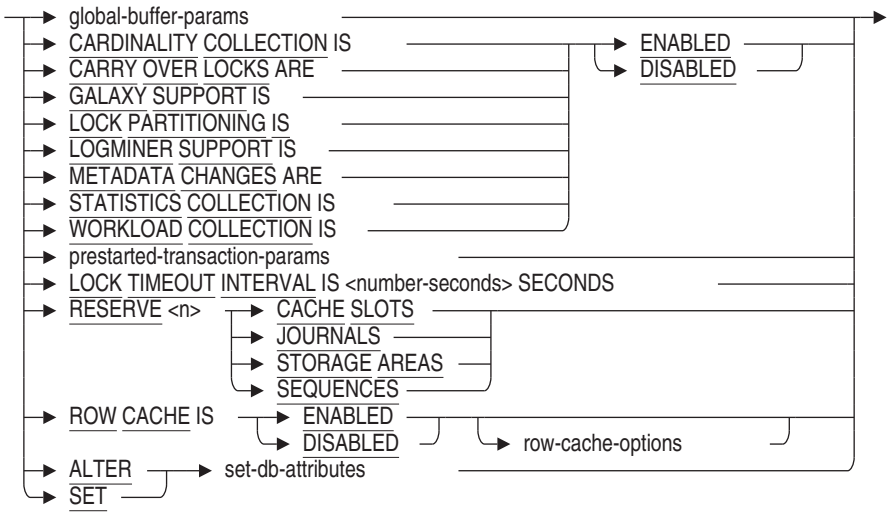
attach-options =



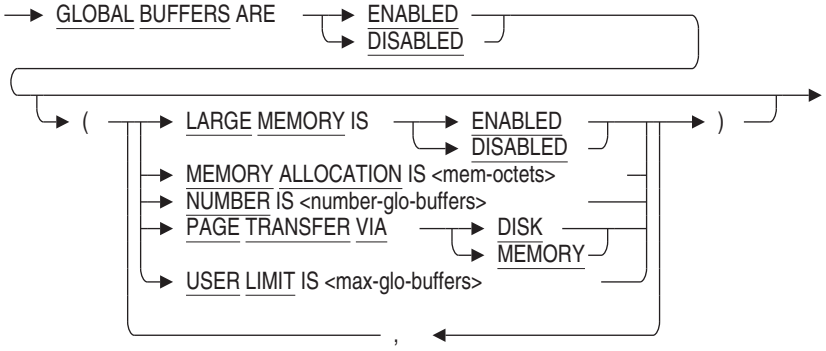
alg-options =



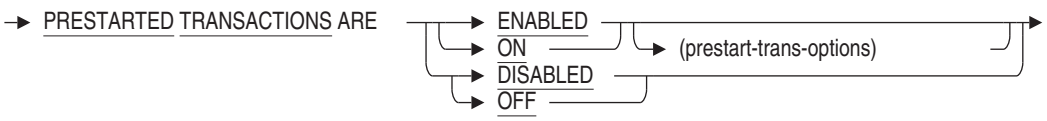
alter-root-file-params2 =



global-buffer-params=



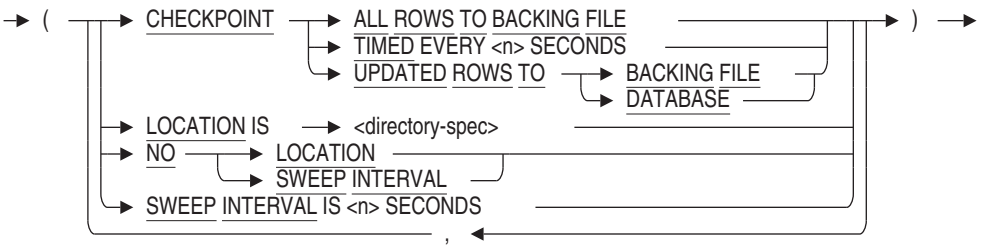
prestarted-transaction-params =



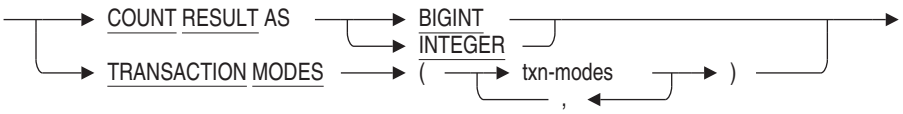
ALTER DATABASE Statement

prestart-trans-options =

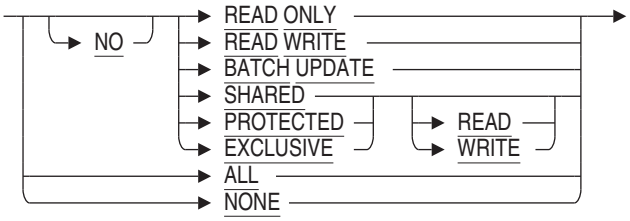
row-cache-options =



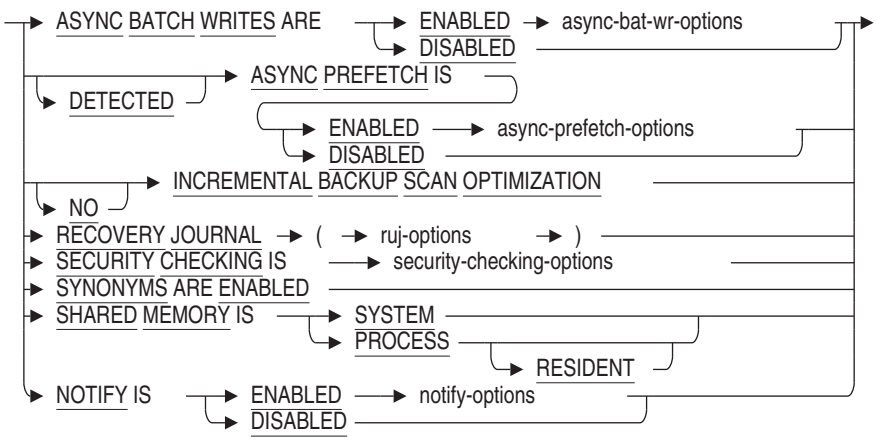
set-db-attributes =



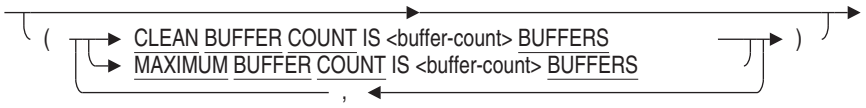
txn-modes =



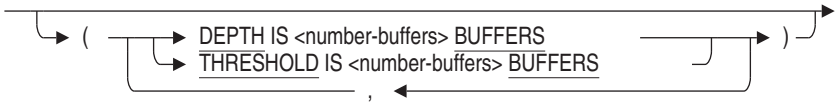
alter-root-file-params3 =



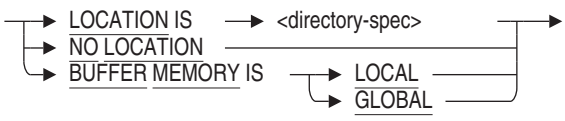
asynch-bat-wr-options =



async-prefetch-options =

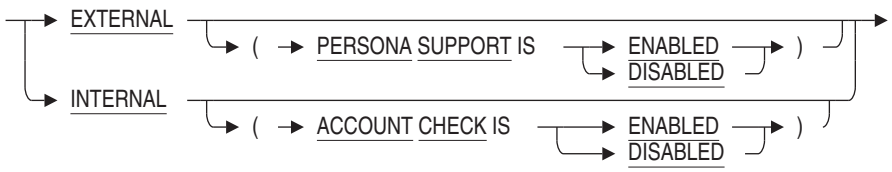


ruj-options =

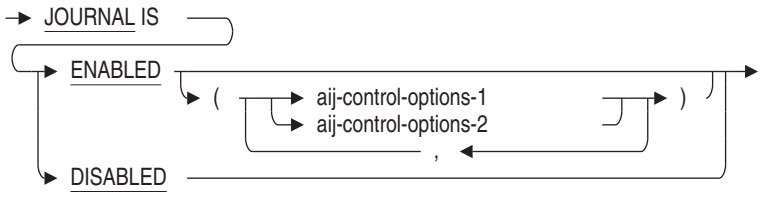


ALTER DATABASE Statement

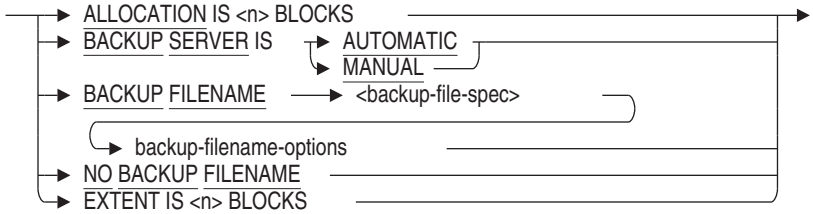
security-checking-options =



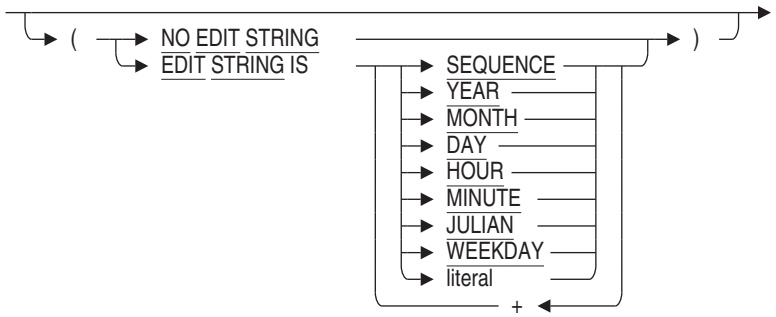
alter-journal-params =



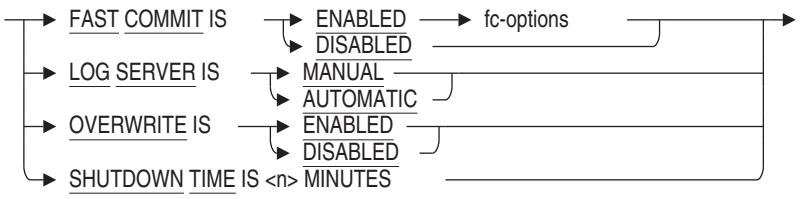
aij-control-options-1 =



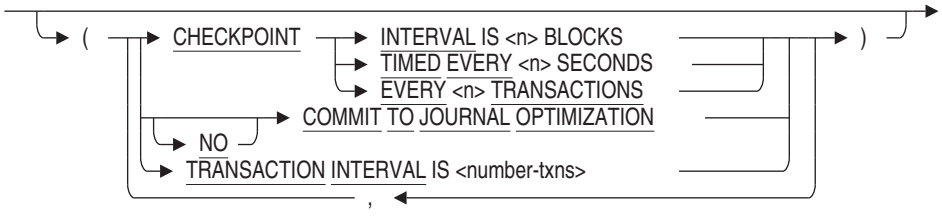
backup-filename-options =



aij-control-options-2 =



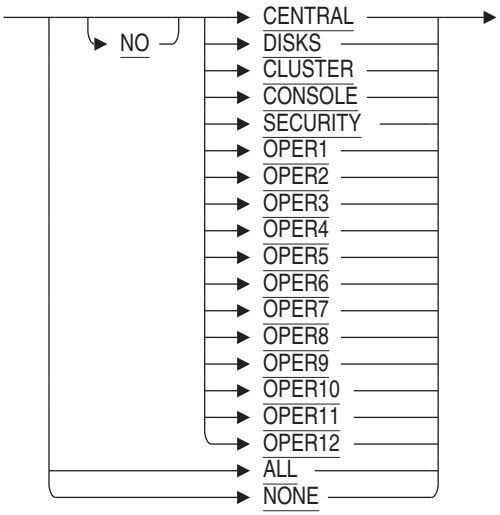
fc-options =



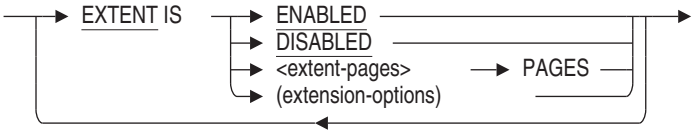
notify-options =

ALTER DATABASE Statement

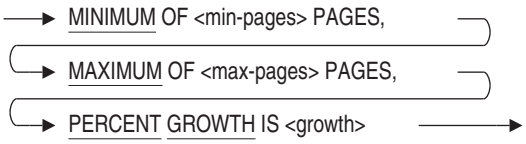
operator-class =



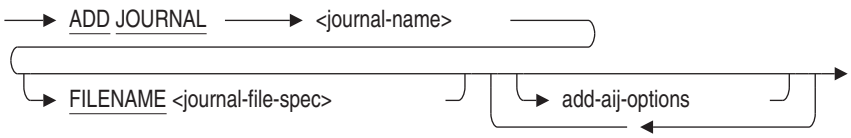
extent-params =



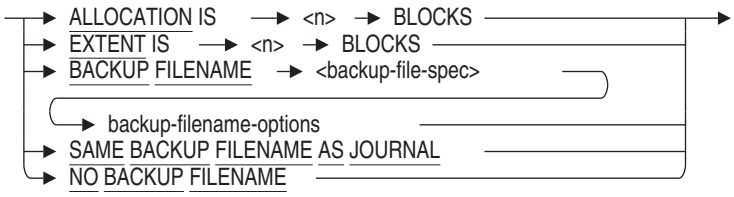
extension-options =



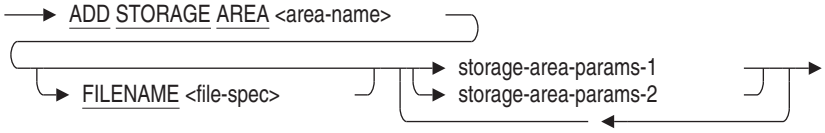
add-journal-clause =



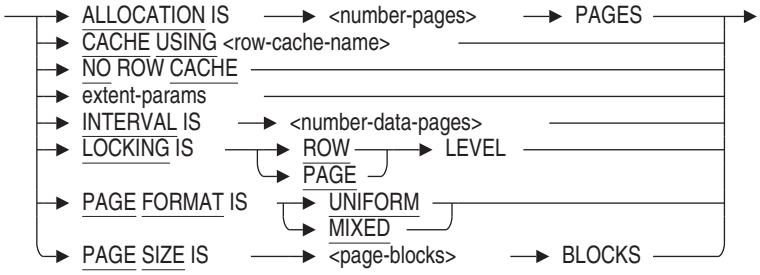
add-aij-options =



add-storage-area-clause =

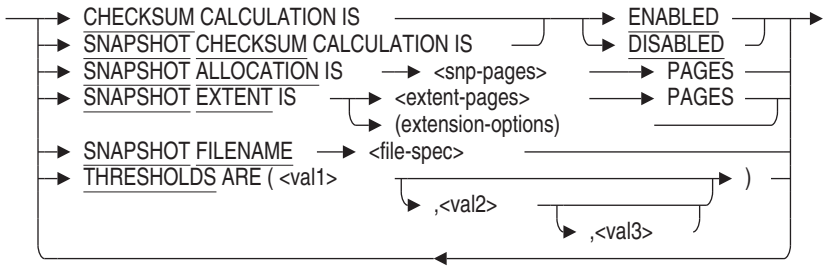


storage-area-params-1 =



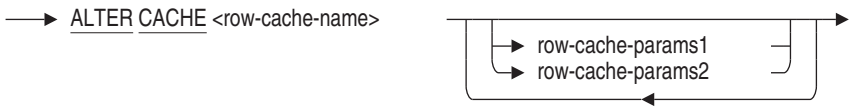
ALTER DATABASE Statement

storage-area-params-2 =

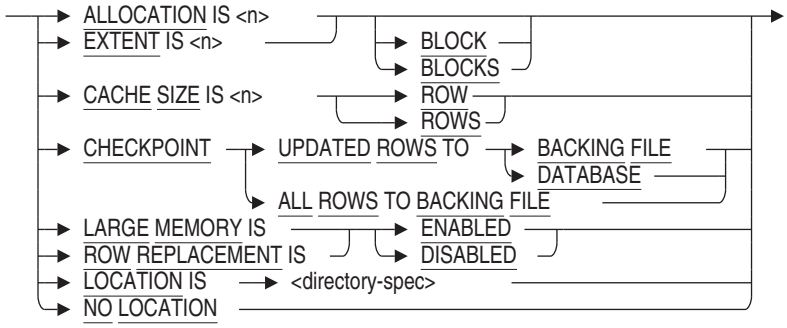


add-row-cache-clause =

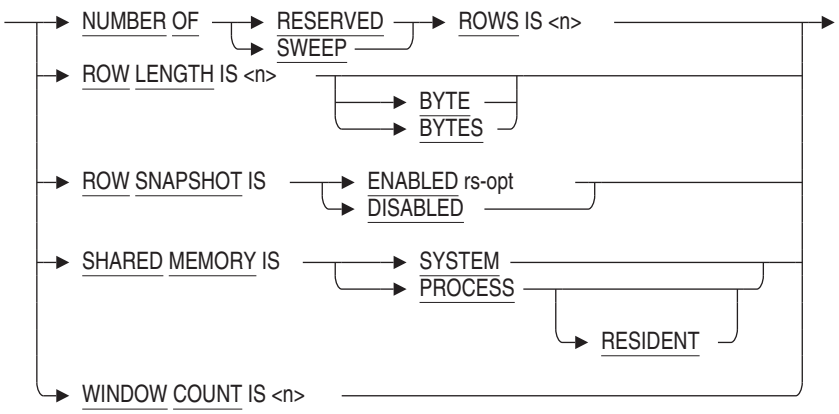
alter-row-cache-clause =



row-cache-params1 =

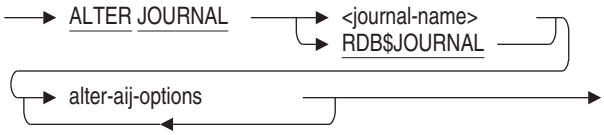


row-cache-params2 =

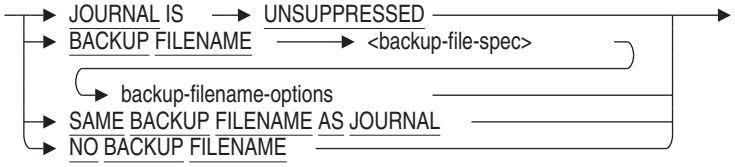


rs-opt =

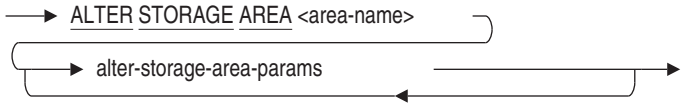
alter-journal-clause =



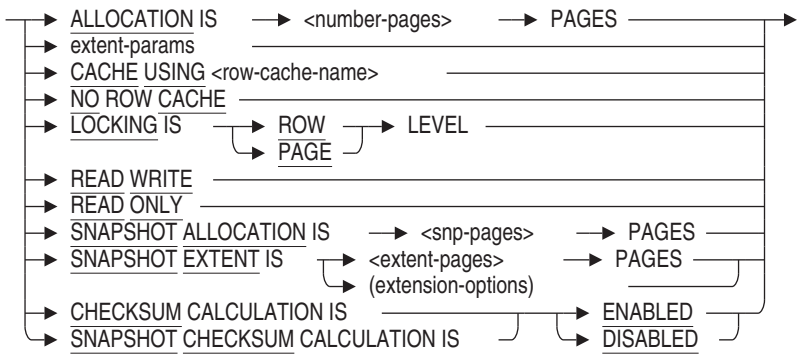
alter-aij-options =



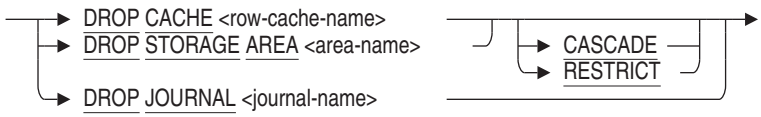
alter-storage-area-clause =



alter-storage-area-params =



drop-clause =



Arguments

ADD CACHE row-cache-name

Adds a new row cache. For information regarding the row-cache-params-1 and row-cache-params-2, see the descriptions under the CREATE CACHE clause.

ADD JOURNAL journal-name

Creates a new journal file.

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
- 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 creates 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.

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, 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.

ALERT OPERATOR
Specifies which operator will be notified of the occurrence of a database system 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.

Operator Class	Meaning
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.
[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] CONSOLE	The CONSOLE class broadcasts messages to the Oracle Enterprise Manager (OEM). NO CONSOLE inhibits broadcast to OEM.
[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.

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 Rdb Guide to Database Design and Definition*.

ALLOCATION IS number-pages PAGES
The number of database pages allocated to the storage area. Rdb will automatically extend this allocation to account for internal structure pages, such as SPAM (spage management) pages.

The altered area is extended if the specified value exceeds the current area allocation. Otherwise the specified value is ignored with a warning as shown in this example.

```
SQL> alter database filename mf_personnel alter storage area JOBS allocation
is 400 pages;
%RDMS-W-NOTEXTENDED, area BLUGUM_1:[SMITHI.DATABASES.V72]JOBS.RDA;1 cannot
be extended to 400 pages
```

ALTER CACHE row-cache-name
Alters an existing row cache.

ALTER COUNT RESULT
SET COUNT RESULT
By default, Rdb returns the result of the COUNT(*), COUNT (DISTINCT expression), and COUNT(expression) as a BIGINT type. The clause can be used to instruct SQL to fetch an INTEGER type result instead.

In most cases this change is unnecessary as any application that fetches a COUNT into an integer host variable will have the value automatically converted to the target data type. However, some applications expect an integer type (such as those written in Dynamic SQL) so this clause sets a permanent database attribute to ensure correct application behavior.

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

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.

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

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 CALCULTION
- SNAPSHOT CHECKSUM CALCULATION

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

- LOCKING IS PAGE LEVEL
- READ WRITE
- READ ONLY

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.

ALTER TRANSACTION MODES

Enable or disable transaction modes. This is an offline operation and requires exclusive database access.

For example, if the current transaction modes are SHARED and READ ONLY and you want to allow only EXCLUSIVE mode, use the following statement:

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

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.

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.

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).

Edit String Option	Meaning
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.

BACKUP SERVER IS AUTOMATIC
BACKUP SERVER IS MANUAL
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.

BUFFER SIZE IS buffer-blocks BLOCKS
Specifies the number of blocks Oracle Rdb allocates per buffer. You need to specify an unsigned integer greater than zero. The default buffer size is 4 times the PAGE SIZE value (16 blocks for the default PAGE SIZE of 4).

The buffer size is a global parameter and the number of blocks per page (or buffer) is constrained to 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.

CACHE USING row-cache-name
Specifies that the named row cache is the default physical row cache for all storage areas in the database. All rows stored in each storage area are cached, regardless of whether they consist of table data, segemented string data, or are special rows such as index nodes.

You must either add the specified cache before completing the ALTER DATABASE statement, or it must already exist.

Alter the database and storage area to asign a new physical area row cache that overrides the database default physical area row cache. Only one physical area row cache is allowed for each storage area.

You can have multiple row caches that contain rows for a single storage area by defining logical area row caches, where the row cache name matches the name of a table or index.

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

CARDINALITY COLLECTION IS ENABLED
CARDINALITY COLLECTION IS DISABLED
Specifies whether or not the optimizer records cardinality updates in the system tables. When enabled, the optimizer collects cardinalities for tables and 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 logical area locks (table and index) 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.

CHECKPOINT EVERY n TRANSACTIONS

A FAST COMMIT option which allows the checkpoint to be generated after a set number of transactions.

See the following example.

```
SQL> alter database
cont>     filename db$:scratch
cont>
cont>     journal is enabled
cont>         (fast commit is enabled
cont>             (checkpoint every 20 transactions,
cont>              checkpoint timed every 20 seconds
cont>             )
cont>         )
cont>     add journal rdb$journal
cont>         filename db$:scratch_aij
cont> ;
%RDMS-W-DOFULLBCK, full database backup should be done to ensure future
recovery
```

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 write modified pages to disk. 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.

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 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>    );
```

CHECKPOINT UPDATED ROWS TO BACKING FILE
CHECKPOINT UPDATED ROWS TO DATABASE
CHECKPOINT ALL ROWS TO BACKING FILE

Specifies the default source and target during checkpoint operations for all row caches. If ALL ROWS is specified, then the source records written during each

checkpoint operation are both the modified and the unmodified rows in a row cache. If UPDATED ROWS is specified, then just the modified rows in a row cache are checkpointed each time.

If the target of the checkpoint operation is BACKING FILE, then the RCS process writes the source row cache entries to the backing (.rdc) files. The row cache LOCATION, ALLOCATION, and EXTENT clauses are used to create the backing files. Upon recovery from a node failure, the database recovery process is able to repopulate the row caches in memory from the rows found in the backing files.

If the target is DATABASE, then updated row cache entries are written back to the database. The row cache LOCATION, ALLOCATION, and EXTENT clauses are ignored. Upon recovery from a node failure, the database recovery process has no data on the contents of the row cache. Therefore, it does not repopulate the row caches in memory.

The CHECKPOINT clause of the CREATE CACHE, ADD CACHE, or ALTER CACHE clause overrides this database-level CHECKPOINT clause.

CHECKSUM CALCULATION
SNAPSHOT CHECKSUM CALCULATION

This option allows you to enable or disable calculations of page checksums when pages are read from or written to the storage area or snapshot files.

The default is ENABLED.

Note

Oracle Corporation 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 Corporation 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 Corporation 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 Corporation 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.

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

The default is five buffers. The minimum value is one; 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. For information about how to set the values, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

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.

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.

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.

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 asynchronously fetches pages further in the sequence. This avoids wait times when the page is really needed.

Detected asynchronous prefetch is enabled by default.

DICTIONARY IS REQUIRED
DICTIONARY IS NOT REQUIRED
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.

DICTIONARY IS USED
DICTIONARY IS NOT USED
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.

DROP CACHE row-cache-name CASCADE
DROP CACHE row-cache-name RESTRICT
Deletes the specified row cache from the database. If the mode is RESTRICT, then an exeption is raised if the row cache is assigned to a storage area. If the mode is CASCADE, then the row cache is removed from all referencing storage areas.

The default is RESTRICT if no mode is specified.

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.

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.

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.

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.

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 specifying CHECKPOINT EVERY *n* TRANSACTIONS.
- A specified time interval elapsed. You set this by specifying the CHECKPOINT TIMED EVERY *n* SECONDS clause.
- The after-image journal (.aij) file increased by a specified number of blocks. 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.

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.

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.

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.

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

GALAXY SUPPORT IS ENABLED
GALAXY SUPPORT IS DISABLED
Allows global memory to be shared in an OpenVMS Galaxy configuration. Galaxy support is disabled by default.

OpenVMS Galaxy is a software architecture for the OpenVMS Alpha operating system that enables multiple instances of OpenVMS to execute cooperatively in a single computer. An instance refers to a copy of the OpenVMS Alpha operating system. As an extension of the existing OpenVMS cluster support within Oracle Rdb, Oracle Rdb provides support for databases opened on multiple instances (or nodes) within a Galaxy system to share data structures in memory. Within an Oracle Rdb Galaxy environment, all instances with an open database share:

- Database root objects (for example, TSN blocks and SEQ blocks)
- Global buffers (if enabled)
- Row caches and Row Cache Server process (RCS) (if enabled)

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 attach (GLOBAL BUFFERS ARE DISABLED). For more than one attach 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 attach at the same time, although only one attach 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

If GALAXY SUPPORT is enabled, then a single global buffer pool is shared by all Galaxy nodes.

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.

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.

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.

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.

LOCATION IS directory-spec
Specifies the name of the default directory to which row cache backing file information is written. The database system generates a file name (row-cache-name.rdc) automatically for each row cache backing file it creates when the RCS process starts up. Specify a device name and directory name enclosed within single quotation marks (′); do not include a file specification. The file name is the row-cache-name specified when creating the row cache. By default, the location is the directory of the database root file.

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

This clause is ignored if the row cache is defined to checkpoint to the database.

LOCK PARTITIONING IS ENABLED
LOCK PARTITIONING IS DISABLED
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.

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 to 30, SQL still uses the interval 25. For more information on timeout intervals, see the *Oracle Rdb7 Guide to Distributed Transactions*.

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.

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 because these page locks are held until COMMIT/ROLLBACK time. 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 and long transactions may use excessive locks.

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

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

LOG SERVER IS MANUAL
LOG SERVER IS AUTOMATIC
Specifies if the AIJ log server (ALS) 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*.

LOGMINER SUPPORT IS ENABLED
LOGMINER SUPPORT IS DISABLED
Allows additional information to be written to the after-image journal file to allow the use of the RMU Unload After_Image command. See the *Oracle RMU Reference Manual* for more details. Logminer support is disabled by default.

The LOGMINER SUPPORT clause allows the continuous mode for LogMiner to be enabled and disabled.

- LOGMINER SUPPORT IS ENABLED (CONTINUOUS)

  Enables continuous LogMiner.

- LOGMINER SUPPORT IS ENABLED (NOT CONTINUOUS)

  Disables continuous LogMiner, but leaves LogMiner enabled.
- LOGMINER SUPPORT IS DISABLED

  Disables LogMiner, including disabling continuous LogMiner.

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. For information about how to set the values, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

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

MEMORY ALLOCATION IS mem-octets
Allows the database administrator to define the size of the GLOBAL BUFFER pool using direct memory sizing as an alternate to specifying the NUMBER of pages. The value of mem-octets is an unsigned numeric literal or a quantified numeric literal. This clause is not compatible with the NUMBER IS clause. Use just one of the keywords to define the size of the global buffers.

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.

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

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 2.2.11.

NO BACKUP FILENAME
Removes a previously established backup file specification.

NO LOCATION
This is a subclause of other clauses and has different effects, depending upon the clause in which it is used, as follows:

- In the row-cache-options clause

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

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

- In a CREATE CACHE, ADD CACHE, or ALTER CACHE clause (row-cache-params1 clause)

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

This clause is ignored if the row cache is defined to checkpoint to the database.

NO ROW CACHE
Specifies that the database default is to not assign a row cache 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 to override the database default. Only one row cache is allowed for each storage area.

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

NO SWEEP INTERVAL
NO SWEEP INTERVAL disables periodic timed sweeps.

NOTIFY IS ENABLED
NOTIFY IS DISABLED
Specifies whether system notification is enabled or disabled.

When the system notification is enabled, the system is notified (using the OpenVMS OPCOM facility) in the event of events such as running out of disk space for a journal.

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 replaces any operator classes set by the RMU Set After_Journal Notify command.

The default is disabled.

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. 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.

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 524,288. The default is 250 buffers.

NUMBER OF CLUSTER NODES IS number-nodes (SINGLE INSTANCE)
NUMBER OF CLUSTER NODES IS number-nodes (MULTIPLE INSTANCE)
Sets the upper limit on the maximum number of VMS cluster 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 VMS cluster node limit set by your system administrator.

The Oracle Rdb root file data structures (.rdb) are mapped to shared memory, each such shared memory copy is known as an Rdb instance. When there is only one copy of shared memory containing root file information, several optimizations are enabled to reduce locking and root file I/O. activity. Specify NUMBER OF CLUSTER NODES is set to 1, or use the SINGLE INSTANCE clause to enable these optimizations.

MULTIPLE INSTANCE means that the Oracle Rdb root file data structures are mapped on different system and are kept consistent through disk I/O. Such systems can not benefit from single instance optimizations. MULTIPLE INSTANCE is the default.

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).

You can specify any number greater than or equal to 2 and less than or equal to 524,288. The default value for the NUMBER OF RECOVERY BUFFERS parameter is 250. 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.

You can override the default number of database recovery buffers by defining a value for the system logical name RDM$BIND_DBR_BUFFER_CNT. This will be used by database detached processes (DBR) when allocating buffers during recovery.

```
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 Rdb Guide to Database Maintenance*.

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 5 users.

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.

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.

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.

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 NODES must be one, or SINGLE INSTANCE must be specified in the NUMBER OF CLUSTER NODES clause.
- 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*.

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

PRESTARTED TRANSACTIONS ARE ENABLED (prestart-trans-options)
Enables the prestarting of transactions.

Note that the keyword ON, available in previous versions, is synonymous with ENABLED.

This clause is used to establish a permanent database setting for prestarted transactions. In prior versions, this clause was only used to temporarily set the mode for prestarted transaction for the implicit attach performed by the CREATE DATABASE and IMPORT DATABASE statements.

The prestart-trans-options can be one of the following clauses:

- WAIT n SECONDS FOR TIMEOUT

  The n represents the number of seconds to wait before aborting the prestarted transaction. Timing out the prestarted transaction may prevent snapshot file growth in environments where servers stay attached to the database with long periods of inactivity.

- WAIT n MINUTES FOR TIMEOUT

  The n represents the number of minutes to wait before aborting the prestarted transaction.

- NO TIMEOUT

  This is the default for a prestarted transaction.

PRESTARTED TRANSACTIONS ARE DISABLED
Disables the prestarting of transactions.

Note that the keyword OFF, available in previous versions, is synonymous with DISABLED.

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*.

RECOVERY JOURNAL (BUFFER MEMORY IS LOCAL)
RECOVERY JOURNAL (BUFFER MEMORY IS GLOBAL)
Specifies whether RUJ buffers will be allocated in global or local memory.

The RUJ buffers used by each process are normally allocated in local virtual memory. With the introduction of row caching, these buffers now can be assigned to a shared global section (global memory) on OpenVMS, so that the recovery process can process this in-memory buffer and possibly avoid a disk access.

You can define this buffer memory to be global to improve row caching performance for recovery. If row caching is disabled, then buffer memory is always local.

RECOVERY JOURNAL (LOCATION IS directory-spec)

Specifies the location, including device and directory, in which the recovery-unit journal (.ruj) file is written. Do not include network node names, file names or process-concealed logical names. The default is the current user's login device.

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

Following is an example using this clause:

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

RECOVERY JOURNAL (NO LOCATION)

Removes a location previously defined by a RECOVERY JOURNAL LOCATION IS clause. This causes the recovery journal to revert to the default location.

RESERVE n CACHE SLOTS

Specifies the number of row caches for which slots are reserved in the database.

You can use the RESERVE CACHE SLOTS clause to reserve slots in the database root file for furture use by the ADD CACHE clause. You can only add row caches if row cache slots are available. Slots become available after you issue a DROP CACHE clause or a RESERVE CACHE SLOTS clause.

You cannot reduce the number of reserved slots for row caching. If you reserve 10 slots and later reserve 5 slots, a total of 15 slots are reserved for row caches.

RESERVE n JOURNALS

Specifies the number of journal files for which slots are to reserve 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 SEQUENCES

Specifies the number of sequences for which slots are reserved in the database. Sequences are reserved in multiples of 32. Thus, if you specify a value less

than 32 for n, 32 slots are reserved. If you specify a value of 33, 64 slots are reserved, and so on.

You can use the RESERVE SEQUENCES clause to reserve slots in the database root file for future use by the CREATE SEQUENCE statement. Sequences can be created only if sequence slots are available. Slots become available after a DROP SEQUENCE statement or a RESERVE SEQUENCES clause of the ALTER DATABASE statement is executed.

RESERVE n STORAGE AREAS
Specifies the number of storage areas for which slots are to reserve 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 the row caching feature is enabled.

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

When row caching is disabled, all previously created and assigned caches remain and will be available if row caching is enabled again.

The following conditions must be true in order to use row caches:

- The number of cluster nodes is one
- After-image journaling is enabled
- Fast commit is enabled

- One or more cache slots are reserved
- Row caching is enabled

Use the RMU Dump Header command to check if you have met the requirements for using row caches. The following command output displays a warning for every requirement that is not met:

```
   .
   .
   .
Row Caches...
- Active row cache count is 0
- Reserved row cache count is 1
- Sweep interval is 1 second
- Default cache file directory is ""
- WARNING: Maximum node count is 16 instead of 1
- WARNING: After-image journaling is disabled
- WARNING: Fast commit is disabled
   .
   .
   .
```

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

SECURITY CHECKING

Traditionally Oracle Rdb has performed security checking using the operating system security layer (for example, the UIC and rights identifiers of the OpenVMS operating system).

The access control list (ACL) information stored in the database contains a granted privilege mask and a set of users represented by a unique integer (for example, a UIC).

There are two modes of security checking:

1. SECURITY CHECKING IS EXTERNAL

   This is the default. External security checking recognizes database users as operating system user identification codes (UICs) and roles as special rights identifiers or groups. PERSONA support is enabled or disabled as follows:

   - SECURITY CHECKING IS EXTERNAL (PERSONA SUPPORT IS ENABLED)

     Enables the full impersonation of an OpenVMS user. This means the UIC and the granted right identifiers are used to check access control list permissions. This is the default for a new database.

   - SECURITY CHECKING IS EXTERNAL (PERSONA SUPPORT IS DISABLED)

     Disables the full impersonation of an OpenVMS user. Only the UIC is used to check access control list permissions.

2. SECURITY CHECKING IS INTERNAL

   In this mode, Oracle Rdb records users (username and UIC) and roles (rights identifiers) in the database. The CREATE USER and CREATE ROLE statements perform this action explicitly, and GRANT will perform this implicitly. This type of database can now be moved to another system and is only dependent on the names of the users and roles.

   - SECURITY CHECKING IS INTERNAL (ACCOUNT CHECK IS ENABLED)

     The ACCOUNT CHECK clause ensures that Oracle Rdb validates the current database user with the user name (such as defined with a CREATE USER statement) stored in the database. This prevents different users with the same name from accessing the database. Therefore, this clause might prevent a breach in security.

     The ACCOUNT CHECK IS ENABLED clause on OpenVMS forces the user session to have the same user name and UIC as recorded in the database.

   - SECURITY CHECKING IS INTERNAL (ACCOUNT CHECK IS DISABLED)

     If you specify the ACCOUNT CHECK IS DISABLED clause, then a user with a matching UIC (also called a profile-id) is considered the same as the user even if his or her user name is different. This allows support for multiple OpenVMS users with the same UIC.

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.

SHARED MEMORY IS SYSTEM
SHARED MEMORY IS PROCESS
SHARED MEMORY IS PROCESS RESIDENT
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 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.

The default is SHARED MEMORY IS PROCESS.

The SHARED MEMORY clause determines whether database root global sections (including global buffers when enabled) or whether the cache global sections are created in system space or process space. The RESIDENT option extends the PROCESS option by making the global section memory resident.

To enable or disable SHARED MEMORY IS PROCESS RESIDENT, the process executing the command must be granted the VMS$MEM_RESIDENT_USER rights identifier. When this feature is enabled, the process that opens the database must also be granted the VMS$MEM_RESIDENT_USER rights identifier. Oracle Corporation recommends using the RMU Open command when utilizing this feature.

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).

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.

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.

In this mode any READ ONLY transaction will be converted to READ WRITE mode automatically.

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. For more information about when to use statistics collection, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

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.

SWEEP INTERVAL IS n SECONDS
Specifies the interval, in seconds, between each Record Cache Server (RCS) sweep. Allowable values must be in the range from 1 second to 3600 seconds (1 hour). The default is 1.

The **Record Cache Server (RCS)** is a detached server process automatically invoked by the monitor when row caching is active.

A **sweep** is one full pass through all active row cache areas to write modified rows back to the database storage areas.

SYNONYMS ARE ENABLED
Adds the optional system table RDB$OBJECT_SYNONYMS that is used for the CREATE SYNONYM, ALTER . . . RENAME TO and RENAME statements. The default if omitted is disabled.

THRESHOLD IS number-buffers BUFFERS
This number represents the number of sequential buffer accesses that must be detected before prefetching is started. The default is four buffers.

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.

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.

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.
[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 tables to be reserved for shared mode. That is, other users can work with those tables.
[NO] PROTECTED [READ | WRITE]	Allows tables to be reserved for protected mode. That is, other users can read from those tables.
[NO] EXCLUSIVE [READ | WRITE]	Allows tables to be reserved for exclusive access. That is, other users are prevented access to those tables, even in READ ONLY transactions.
ALL	Allows other users to work with all tables.
NONE	Allows no access to tables.

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

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 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.

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'
USING ('password', 'password')
A character string literal that specifies the user's password for the user name specified in the USER clause. If the user requires two passwords, then specify both values in parenthesis and separated by a comma.

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.

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 Optimizer-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.

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 Rdb 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 LIST STORAGE AREA, or the RDB$SYSTEM storage area.
- When the LOCKING IS PAGE LEVEL or LOCKING IS ROW LEVEL clause is specified at the database level (using the ALTER DATABASE or CREATE DATABASE statements), all storage areas are affected (with the exception of RDB$SYSTEM and the DEFAULT storage area, which is always set to row-level locking).

- 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 Corporation 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 Rdb Guide to Database Design and Definition* for a complete discussion of when to use the IMPORT, EXPORT, and ALTER DATABASE statements.
- Table 6–1 shows 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 Rdb 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 CREATE DROP	Yes	You cannot drop a catalog when there are active transactions that access the catalog.
Collating sequences ALTER CREATE DROP	Yes	You cannot drop a collating sequence if the database or domain in the database uses that collating sequence.
Constraints ALTER CREATE DROP RENAME	Yes	You cannot drop a constraint when there are active transactions that access the tables involved.
Domains ALTER CREATE DROP RENAME	Yes	You cannot alter a domain if stored routines use the domain.
External routines ALTER CREATE DROP RENAME	Yes	Refers to external procedures and functions.
Indexes CREATE ALTER DROP RENAME	Yes	You cannot disable an index or delete an index definition when there are active transactions that access the tables involved.
Modules CREATE DROP ALTER RENAME	Yes	Modules contain stored procedures and functions.
Outlines CREATE DROP ALTER RENAME	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
Profiles	Yes	
ALTER		
CREATE		
DROP		
RENAME		
Protection	Yes	Granting or revoking a privilege takes effect after the user detaches and attaches to the database again.
GRANT		
REVOKE		
Schemas	Yes	You cannot drop a schema when there are active transactions that access the schema.
CREATE		
DROP		
Roles	Yes	
CREATE		
DROP		
ALTER		
RENAME		
Storage areas		
RESERVE	No	This change is not journaled.
CREATE	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.
ADD		
DROP		
ALTER	See comments	You can modify many of the storage area parameters. See Table 6–2 for specific information.
Storage maps	Yes	
ALTER		
CREATE		
DROP		
Sequences	Yes	
ALTER		
CREATE		
DROP		
RENAME		

[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
Synonyms ALTER CREATE DROP RENAME	Yes	
Tables ALTER CREATE RENAME DROP TRUNCATE	Yes	You cannot drop or truncate a table definition when there are active transactions that use the table.
Triggers ALTE CREATE DROPR RENAME	Yes	You cannot delete a trigger definition when there are active transactions that use the trigger or that refer to the tables involved.
User ALTER CREATE DROP RENAME	Yes	
Views CREATE DROP RENAME	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 ALTER	See comments	You can modify many of the database parameters, including storage area parameters. See Table 6–2 for specific information.
CREATE 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.

[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 Rdb 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 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 Database-Wide Parameters While Users Are Attached to the Database

Metadata Update	On Line [1]	Comments
Root File Parameters		
Galaxy Support	No	
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 asyn-chronous 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 sequences	No	
Reserve row cache slots	No	This change is not journaled.
Row cache enabled or disabled	No	
Create or add 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 Database-Wide Parameters While Users Are Attached to the Database

Metadata Update	On Line [1]	Comments
Root File Parameters		
Alter row cache	No	
Delete row cache	No	
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	
Logminer support	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	

[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 Database-Wide Parameters While Users Are Attached to the Database

Metadata Update	On Line [1]	Comments
Root File Parameters		
Journal checkpoint interval	No	
Journal checkpoint time	No	
Journal commit to journal optimization	No	
Journal transaction interval	No	
Journal log server	Yes	
Journal overwrite	Yes	
Journal shutdown time	Yes	
Storage Area Parameters		
Reserve storage area	No	This change is not journaled.
Read or write attribute	Yes	Requires exclusive access to the area.
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 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.
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.

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.

- 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 Rdb 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
```

- 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
- The ADD CACHE automatically assigns the cache to any table or index of the same name. You must use the CACHE USING clause with the ADD STORAGE AREA or ALTER STORAGE AREA clauses of the ALTER DATABASE statement to assign the cache to a storage area.
- The product of the CACHE SIZE and the ROW LENGTH settings determines the amount of memory required for the row cache (some additional overhead and rounding up to page boundaries is performed by the database system). Use RMU to display the required memory.
- A row cache 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 or SINGLE INSTANCE must be specified in the NUMBER OF CLUSTER NODES clause.

- When you alter the row length of a row cache 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.
- 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.
- The number of reserved slots for sequences cannot be decreased.
- If you do not specify the RESERVE n CACHE SLOTS clause, the default number of cache slots is 32.
- The RDB$PROFILES system relation is used to record users and roles created with the CREATE USER and CREATE ROLE statements. When a database is created, the creator is automatically added as a user.
- The PRESTARTED TRANSACTION attribute in the database will be used unless overridden by the RDMS$BIND_PRESTART_TXN logical name, or the PRESTARTED TRANSACTION clause on an explicit ATTACH, CONNECT, or DECLARE ALIAS statement. However, the time value specified for the database is used if prestarted transactions are enabled.
- To enable or disable Galaxy support, the process executing the command must hold the SHMEM privilege. When Galaxy is enabled, the process that opens the database must have the SHMEM privilege enabled. Oracle Corporation recommends using the RMU Open command when utilizing this feature.
- The following usage notes apply to the DROP STORAGE AREA CASCADE clause:
  - 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 part of 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.
  - An index that is not partitioned and resides entirely in the storage area being dropped will be deleted using CASCADE semantics (and therefore will invalidate any query outlines that refer to that index).

– The NOT NULL, PRIMARY KEY, and UNIQUE constraints for affected tables are ignored by the DROP STORAGE AREA CASCADE clause because validation of these constraints is not necessary.

  These types of constraints are not affected by removal of rows from the table. This can save considerable I/O and elapsed time when you perform a DROP STORAGE AREA CASCADE operation. However, CHECK and FOREIGN constraints on the affected table and referencing tables are still evaluated.

– When DROP STORAGE AREA CASCADE is executing, it logs debugging messages to the standard output device or the RDMS$DEBUG_FLAGS_OUTPUT log file on OpenVMS.

  You can enable logging of the debug messages using the logical name RDMS$SET_FLAGS, which accepts the same input as the SQL SET FLAGS statement. For example:

```
$ DEFINE RDMS$SET_FLAGS "STOMAP_STATS, INDEX_STATS, ITEM_LIST"
```

  The SET FLAGS OPTIONS shown in the preceding example enables the following debug output:

  * STOMAP_STATS displays the processing of storage maps for any tables that refer to the dropped storage area.
  * INDEX_STATS displays the processing of any indices which reference the dropped storage area,
  * ITEM_LIST displays the names of any constraints that require processing.

  The output includes the discovered tables and indexes, some decision-point information (does an index need to be deleted, does a partition need to be scanned, and so on), and I/O statistics for the storage map pruning operations.

  Part of the DROP STORAGE AREA CASCADE operation may include deleting tables and indexes. These are processed internally as DROP TABLE CASCADE and DROP INDEX CASCADE operations. However, by the time these commands execute, all references to the dropped storage area will have been removed. Therefore, in many cases the DROP TABLE or DROP INDEX statement only cleans up the metadata definition; there is no need to scan the storage area.

– The time required to delete a storage area file depends on the size of the directory file, the file allocation, and the number of extents made by the file system to grow the file. If the ERASE ON DELETE attribute is enabled on the disk, then this must also be factored into the time

calculations (allow time for the file system to overwrite the file with an erase pattern).

– Note that the read/write I/O statistics are output only if the database has statistics collection enabled. Statistics collection might be disabled if the logical name RDM$BIND_STATS_ENABLED has been set to 0, of if an ALTER DATABASE...STATISTICS COLLECTION IS DISABLED statement has been issued.

- By default, notification of area extends is disabled. To enable this notification, define the logical name RDM$BIND_NOTIFY_STORAGE_AREA_EXTEND to 1. This will result in storage area extend events being notified. The message will be sent to CENTRAL and CLUSTER operators, as well as any other operators named in the NOTIFY IS ENABLED clause.

  The logical name RDM$BIND_NOTIFY_STORAGE_AREA_EXTEND should be defined system-wide so that any process extending an area will cause the notification to occur.

- The ALTER DATABASE statement performs two classes of functions:

  – Changing the database root structures in the .RDB file

  – Modifying the system metadata in the RDB$SYSTEM storage area.

  The first class of changes do not require a transaction to be active. However, the second class does require that a transaction be active. Oracle Rdb does not currently support the mixing of these two classes of ALTER DATABASE clauses.

  When you mix clauses that fall into both classes, the following error message is displayed, and the ALTER DATABASE statement fails.

```
DDLDONOTMIX the {SQL-syntax} clause can not be used with some ALTER DATABASE clauses
```

  For example:

```
SQL> alter database filename MF_PERSONNEL
cont> dictionary is not used
cont> add storage area JOB_EXTRA filename JOB_EXTRA
RDB-F-BAD_DPB_CONTENT, invalid database parameters in the database parameter
%RDMS-E-DDLDONOTMIX, the "DICTIONARY IS NOT USED" clause can not be used with
some ALTER DATABASE clauses
```

  The following clauses modify system metadata, but may not appear with other clauses such as ADD STORAGE AREA or ADD CACHE:

  – CARDINALITY COLLECTION IS { ENABLED | DISABLED }

[1] You may not specify both SYNONYMS ARE ENABLED and MULISCHEMA IS ON for the same database.

- DICTIONARY IS [NOT] REQUIRED
- DICTIONARY IS NOT USED
- METADATA CHANGES ARE { ENABLED | DISABLED }
- MULTISCHEMA IS { ON | OFF } [1]
- SECURITY CHECKING IS { INTERNAL | EXTERNAL }
- SET COUNT RESULT AS { INTERNAL | EXTERNAL }
- SYNONYMS ARE ENABLED [1]
- WORKLOAD COLLECTION IS { ENABLED | DISABLED }

If the DDLDONOTMIX error is displayed, then restructure the ALTER DATABASE into two statements, one for each class of actions.

```
SQL> alter database filename MF_PERSONNEL
cont> dictionary is not used;
SQL> alter database filename MF_PERSONNEL
cont> add storage area JOB_EXTRA filename JOB_EXTRA;
```

- A node specification may only be specified for the root FILENAME clause of the ALTER DATABASE statement.

  This means that the directory or file specification specified with the following clauses can only be a device, directory, file name, and file type:

  - LOCATION clause of the ROW CACHE IS ENABLED, RECOVERY JOURNAL, ADD CACHE, and CREATE CACHE clauses
  - SNAPSHOT FILENAME clause
  - FILENAME and SNAPSHOT FILENAME clauses of the ADD STORAGE AREA and CREATE STORAGE AREA clauses
  - BACKUP FILENAME clause of the JOURNAL IS ENABLED, ADD JOURNAL, and ALTER JOURNAL clauses
  - BACKUP SERVER and CACHE FILENAME clauses of the JOURNAL IS ENABLED clause
  - FILENAME clause of the ADD JOURNAL clause

- Very large memory (VLM) allows Oracle Rdb to use as much physical memory as is available on your system and to dynamically map it to the virtual address space of database users. VLM provides access to a large amount of physical memory through small virtual address windows. Even though VLM is defined in physical memory, the virtual address windows are defined and maintained in each user's private virtual address space.

Global buffers in VLM are fully resident, or pinned, in memory and do not directly affect the quotas of the working set of a process.

The number of virtual address "windows" per process is based on the global buffers maximum 'allocate set' parameter specified with the USER LIMIT IS value of the ALTER DATABASE . . . GLOBAL BUFFERS command and is not directly adjustable.

The LARGE MEMORY parameter of the ALTER DATABASE . . . GLOBAL BUFFERS command is used to specify that global buffers are to be created in very large memory:

```
SQL> ALTER DATABASE FILENAME 'MYBIGDB.RDB'
cont> GLOBAL BUFFERS ARE ENABLED
cont>   (NUMBER IS 250000,
cont>    USER LIMIT IS 500,
cont>    LARGE MEMORY IS ENABLED);
```

It is important that you consider the amount of memory available on your system before you start using VLM for global buffers. On OpenVMS systems, you can use the DCL command SHOW MEMORY/PHYSICAL to check the availability and usage of physical memory. This command displays information on how much memory is used and how much is free. The free memory is available for VLM global buffers in addition to user applications.

The total number of global buffers per database is limited to 524,288 and the maximum buffer size is 64 blocks. This yields a global buffer maximum of 16gb (2,097,152 Alpha pages). This restriction may be relaxed in future releases of Oracle Rdb.

Refer to the *Oracle Rdb7 Guide to Database Performance and Tuning* for additional information about the global buffers feature.

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: 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 3: 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;
```

Example 4: Reserving Slots for Sequences

This example shows that reserving extra sequences in the database adds to the existing 32 that are provided by default, and the count rounded up to the next multiple of 32 (that is, 64).

```
$ SQL$ ALTER DATABASE FILENAME MF_PERSONNEL RESERVE 10 SEQUENCES;
%RDMS-W-DOFULLBCK, full database backup should be done to ensure future
recovery
$ RMU/DUMP/HEADER=SEQUENCE MF_PERSONNEL
*-------------------------------------------------------------------------------
* Oracle Rdb V7.1-200                                  15-AUG-2003 14:54:26.55
*
* Dump of Database header
*     Database: USER2:[DOCS.WORK]MF_PERSONNEL.RDB;1
*
*-------------------------------------------------------------------------------

Database Parameters:
    Root filename is "USER2:[DOCS.WORK]MF_PERSONNEL.RDB;1"
    Sequence Numbers...
   .
   .
   .
    Client sequences...
      - 64 client sequences have been allocated
      - 0 client sequences in use
```

Example 5: Adding and Enabling a Row Cache on OpenVMS

The MF_PERSONNEL database is altered to add a row cache, apply it to several storage areas and enable row caching. The example further assumes that after image journals have already been defined for the database, they are required for the JOURNAL IS ENABLED clause to succeed.

```
SQL> /*
***> Prepare the database for ROW CACHE, include extra
***> capacity for later additions
***> */
SQL> alter database
cont>     filename MF_PERSONNEL
cont>     number of cluster nodes is 1
cont>     journal is ENABLED (fast commit is enabled)
cont>     reserve 20 cache slots
cont>     row cache is ENABLED
cont>
cont> /*
***> Create a physical cache for all the employee rows
***> */
cont>  add cache EMPIDS_RCACHE
cont>     shared memory is SYSTEM
cont>     row length is 126 bytes
cont>     cache size is 204 rows
cont>     checkpoint updated rows to database
cont>
cont> /*
***> Apply the cache to each of the relevant storage areas
***> */
cont>  alter storage area EMPIDS_LOW
cont>     cache using EMPIDS_RCACHE
cont>  alter storage area EMPIDS_MID
cont>     cache using EMPIDS_RCACHE
cont>  alter storage area EMPIDS_OVER
cont>     cache using EMPIDS_RCACHE
cont> ;
%RDMS-W-DOFULLBCK, full database backup should be done to ensure future
recovery
```

Example 6: Establishing a Timeout Value for Prestarted Transactions

```
SQL> ALTER DATABASE
cont>   FILENAME SAMPLE
cont>   PRESTARTED TRANSACTIONS ARE ENABLED
cont>      (WAIT 90 SECONDS FOR TIMEOUT)
cont>   ;
```

Example 7: Altering a Database Specifying the SINGLE INSTANCE Option

This example prepares a database to be run in a 4 node GALAXY cluster. The SINGLE INSTANCE clause is used to enable special optimizations that are available because of the galaxy shared memory.

```
SQL> alter database
cont> filename MF_PERSONNEL
cont> galaxy support is ENABLED
cont> number of cluster nodes is 4 (single instance);
```

Example 8: Disabling storage if snapshot rows

The following example demonstrates using SQL to modify the "C1" cache to disable storage of snapshot rows in cache and to modify the "C5" cache to enable storage of snapshot rows in the cache with a snapshot cache size of 12345 rows:

```
SQL> ALTER DATABASE FILE EXAMPLE_DB
cont> ALTER CACHE C1
cont>  ROW SNAPSHOT IS DISABLED;
cont> ALTER CACHE C5
cont>  ROW SNAPSHOT IS ENABLED (CACHE SIZE IS 12345 ROWS);
```

Example 9: Using the SWEEP INTERVAL clause

Here is an example of using the SWEEP INTERVAL clause.

```
SQL> ALTER  DATABASE FILENAME MF_PERSONNEL
cont>     ROW CACHE IS ENABLED (SWEEP INTERVAL IS 100 seconds)
cont> ;
SQL> attach 'filename MF_PERSONNEL';
SQL> show database rdb$dbhandle
Default alias:
    Oracle Rdb database in file MF_PERSONNEL
.
.
.
        Row Cache is Enabled
        Row cache: sweep interval is 100 seconds
        Row cache: No Location
        Row cache: checkpoint updated rows to backing file
.
.
.
```

Example 10: Using the MEMORY ALLOCATION clause when creating global buffers

The MEMORY ALLOCATION clause allows the size specification to be stated using a quantified numeric value. In this example 1M indicates one megabyte of memory.

```
alter database
    filename TEST_DB

    allocation 110k pages
    snapshot allocation 1k pages
    global buffers are disabled (memory allocation 1M)

    create storage area AREA1
        allocation 100k pages
        snapshot allocation 2k pages
;
```

Example 11: 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
    .
    .
    .
```

ALTER DOMAIN Statement

Alters a domain definition.

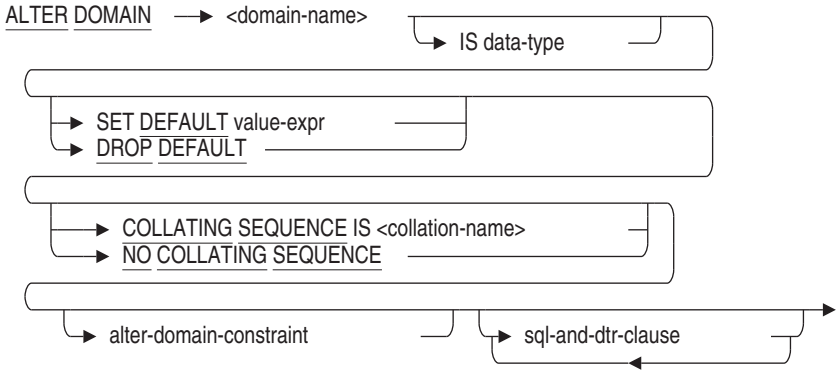
The ALTER DOMAIN statement lets you change the character set, data type, optional default value, optional collating sequence, or optional formatting clauses associated with a domain name. Any table or view definitions that refer to that domain reflect the changes.

Environment

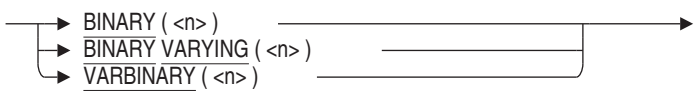
You can use the ALTER DOMAIN statement:

- 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

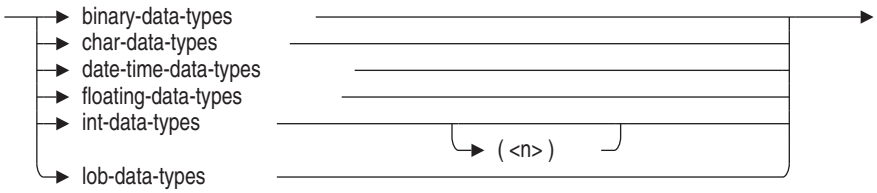
Format



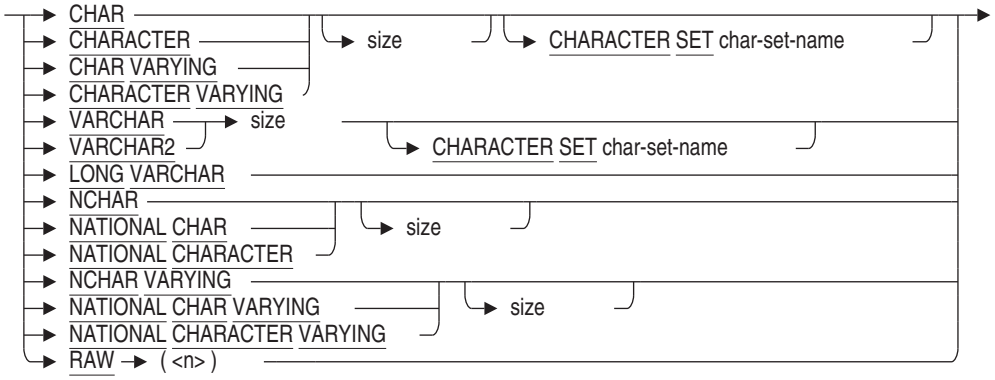
binary-data-types =



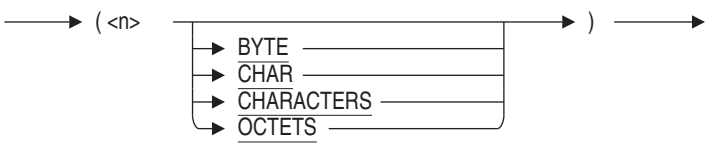
data-type =



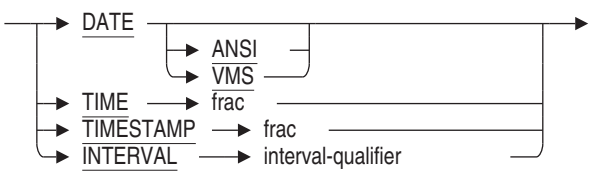
char-data-types =



size =

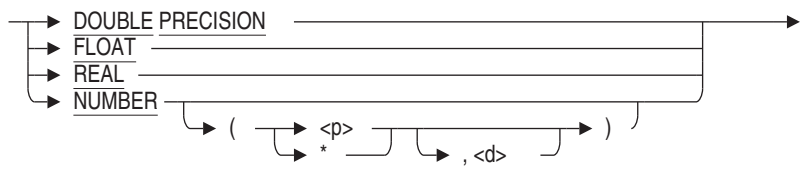


date-time-data-types =

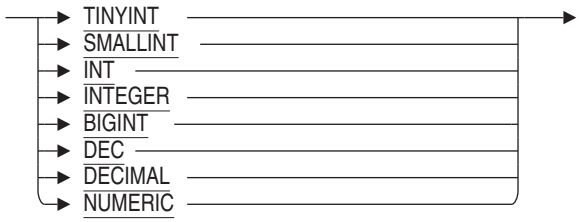


ALTER DOMAIN Statement

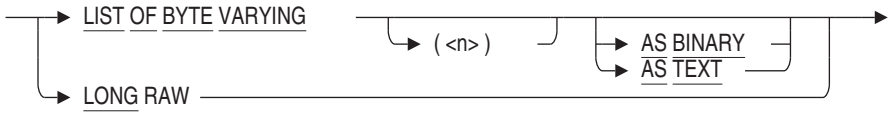
floating-data-types =



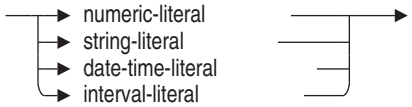
int-data-types =



lob-data-types =

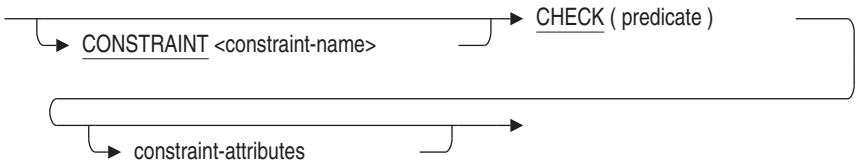


literal =

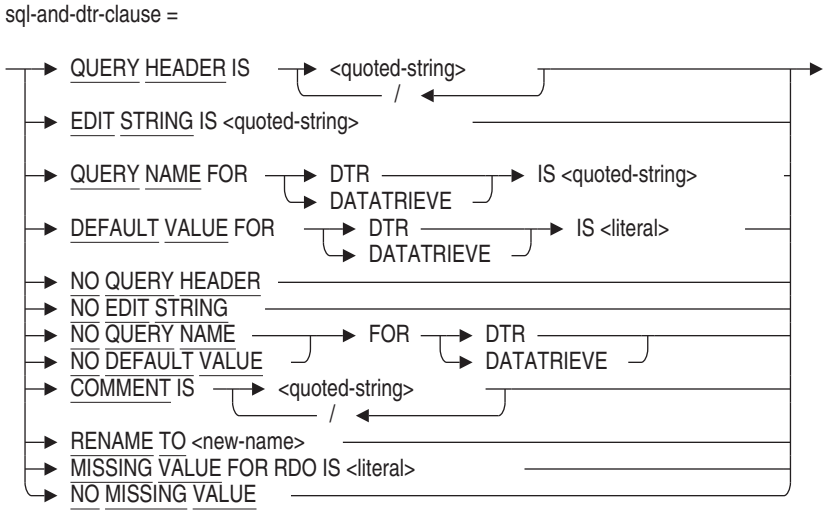


alter-domain-constraint =

domain-constraint =



constraint-attributes =



Arguments

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.

COLLATING SEQUENCE IS collation-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.

COMMENT IS 'string'
Adds a comment about the domain. SQL displays the text of the comment when it executes a SHOW DOMAIN statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

date-time-data-types
A data type that specifies a date, time, or interval. For more information on date-time data types, see Section 2.3.2.

DEFAULT value-expr
Provides a default value for a domain.

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.

For more information about NULL, see Section 2.6.1 and the Usage Notes following this Arguments list.

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.

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

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 SQL99, SQL92, SQL2011, ORACLE LEVEL 1, ORACLE LEVEL 2 or ORACLE LEVEL 3, 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
```

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

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

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

MISSING VALUE FOR RDO IS literal
NO MISSING VALUE
The MISSING VALUE defines a string that is returned when an RDBPRE application fetches a NULL column value. The literal specified for the MISSING VALUE must be compatible with the data type of the domain. Only simple literal expressions are supported; namely DATE VMS literals, character string literals and numeric values (fixed or floating).

NO MISSING VALUE will remove any MISSING VALUE for the domain.

This clause is provided in SQL to allow applications written using the RDBPRE precompilers and databases created using the RDO database definition language to be migrated to SQL-only interfaces and still retain this functionality.

Oracle does not recommend the use of MISSING VALUE for modern applications because these semantics are not defined by the ANSI nor ISO SQL Database Language Standards. MISSING VALUE was introduced for RDO as a way to manage NULL (unknown) values in the database. SQL supports well defined standard functions IS NULL, NULLIF, and COALESCE for such purposes, as well as allowing applications to fetch the null indication into an INDICATOR variable.

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.

RENAME TO
Changes the name of the domain being altered. See the RENAME Statement for further discussion. If the new name is the name of a synonym then an error will be raised.

The RENAME TO clause requires synonyms be enabled for this database. Refer to the ALTER DATABASE Statement SYNONYMS ARE ENABLED clause. Note that these synonyms may be deleted if they are no longer used by database definitions or applications.

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. 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.

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

value-expr
Specifies the default value of a domain.

Usage Notes

- You cannot alter a domain definition unless you have ALTER privilege for 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 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 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 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' statement, you change the length to characters. SET DIALECT also changes the default character length. 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.

- 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.

- 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.
- Using the ALTER DOMAIN statement, when you modify a domain, its attributes are automatically propagated to all referencing tables and views.

  For example, if you modify the data type of a domain, Oracle Rdb updates any view column that refers to that domain, directly or indirectly, to reflect the new attributes of that domain. (A view column can refer indirectly to a domain by using an expression that refers to a base table's column which uses that domain.
- The ALTER DOMAIN ... ADD CONSTRAINT clause performs an implicit ALTER DOMAIN ... DROP ALL CONSTRAINTS prior to applying the new constraint to the domain. Any constraint name defined by prior statements will also be dropped.

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
 VALID IF:  (VALUE > DATE'1900-01-01' OR
                              VALUE IS 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:

```
SQL> create module SAMPLE
cont>     procedure FIRST_NAME
cont>         (in :id id_dom
cont>         ,out :first_name char(40));
cont>     begin
cont>     declare :fn first_name_dom;
cont>     select first_name into :fn
cont>         from employees
cont>         where employee_id = :id;
cont>     -- return capitalized first name
cont>     set :first_name =
cont>         UPPER (substring (:fn from 1 for 1)) ||
cont>         LOWER (substring (:fn from 2));
cont>     end;
cont> end module;
SQL>
SQL> declare :first_name first_name_dom;
SQL> call FIRST_NAME ('00164', :first_name);
 FIRST_NAME
 Alvin
SQL>
SQL> alter domain id_dom
cont>     char(10);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-RTNEXI, field "ID_DOM" is used in routine "FIRST_NAME"
-RDMS-F-FLDNOTCHG, field ID_DOM has not been changed
SQL>
SQL> alter domain first_name_dom
cont>     char(60);
```

- 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.

- 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. Future calls to the stored routine will use the new definition of the domain.

Example 10: Altering a Domain to Provide a Default Value

This examples demonstrates that the default value added to the domain is propagated to the tables using that domain.

```
SQL> -- Display the current domain definition.
SQL> SHOW DOMAIN DEPARTMENT_NAME
DEPARTMENT_NAME                 CHAR(30)
 Comment:        Department name
 Missing Value: None
SQL> -- Alter the domain to provide a default value
SQL> -- for DEPARTMENT_NAME.
SQL> ALTER DOMAIN DEPARTMENT_NAME
cont> SET DEFAULT 'Not Recorded';
SQL> -- Display the altered domain definition.
SQL> SHOW DOMAIN DEPARTMENT_NAME;
DEPARTMENT_NAME                 CHAR(30)
 Comment:        Department name
 Oracle Rdb default: Not Recorded
 Missing Value: None
SQL> -- Insert a record and omit the value for DEPARTMENT_NAME.
SQL> INSERT INTO DEPARTMENTS (DEPARTMENT_CODE)
cont> VALUES
cont> ('GOGO');
1 row inserted
SQL> COMMIT;
SQL> -- Select the newly inserted record to show that the
SQL> -- default for the DEPARTMENT_NAME domain was inserted.
SQL> SELECT * FROM DEPARTMENTS WHERE DEPARTMENT_CODE='GOGO';
 DEPARTMENT_CODE   DEPARTMENT_NAME                 MANAGER_ID
   BUDGET_PROJECTED   BUDGET_ACTUAL
 GOGO              Not Recorded                    NULL
               NULL            NULL
1 row selected
```

ALTER FUNCTION Statement

Allows attributes to be changed for a function that was created using the CREATE MODULE statement or the CREATE FUNCTION statement.

It can be used to:

- Force a stored (SQL) function to be compiled (COMPILE option)
- Modify attributes of external functions
- Change the comment on a function

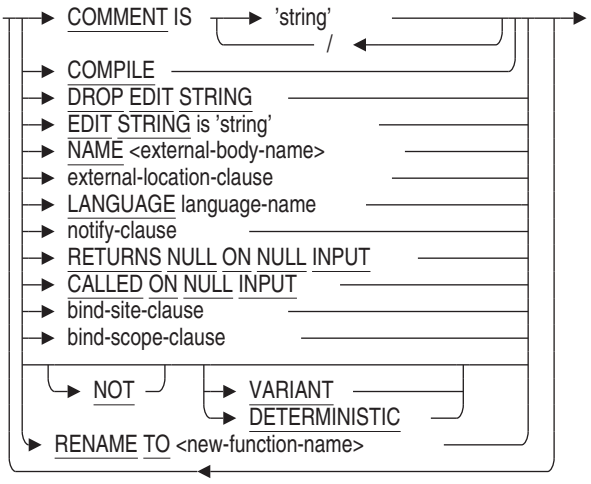
Environment

You can use the ALTER FUNCTION statement:

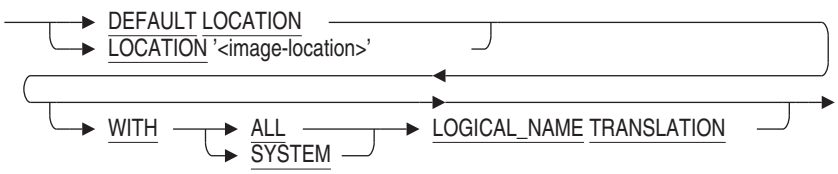
- In interactive SQL
- Embedded in host language programs
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

Format

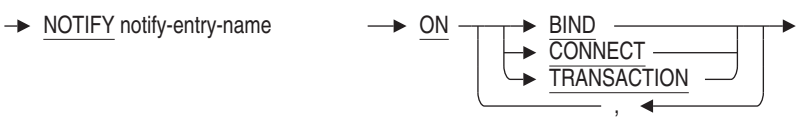
ALTER FUNCTION <function-name>



external-location-clause =

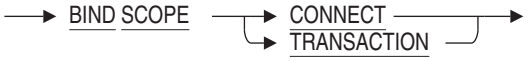


notify-clause =



bind-site-clause =

bind-scope-clause =



Arguments

BIND ON CLIENT SITE
BIND ON SERVER SITE
Selects the execution model and environment for external routine execution.

CLIENT site binding causes the external routine to be activated and executed in the OpenVMS database client (application) process. This is the default binding. This binding offers the most efficient execution characteristics, allows sharing resources such as I/O devices, and allows debugging of external routines as if they were part of the client application. However, this binding may suffer from address space limitations. Because it shares virtual memory with the database buffers, this binding is restricted to the client process system user environment, and prohibits external routine execution in cases of an application running with elevated privileges.

SERVER site binding causes the external routine to be activated in a separate process from the database client and server. The process is started on the same node at the database process. This binding offers reasonable execution characteristics, a larger address space, a true session user environment, and has no restrictions regarding client process elevated privileges. However, this binding does not permit sharing resources such as I/O devices with the client (in particular, there is no connection to the client interactive terminal), and debugging of routines is generally not possible.

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.

COMMENT IS string
Adds a comment about the function. SQL displays the text of the comment when it executes a SHOW FUNCTIONS statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

This clause is equivalent to the COMMENT ON FUNCTION statement.

COMPILE
The COMPILE option forces the Oracle Rdb server to recompile the stored (SQL) function. External functions are not affected.

Use COMPILE when a function has been made invalid by the execution of a DROP . . . CASCADE operation. This mechanism is preferred over the SET FLAGS 'VALIDATE_ROUTINE' method available in previous versions.

DEFAULT LOCATION
LOCATION 'image-location'
A default or specific location for the external routine image. The resulting file specification must include the type *.exe*.

This can be an image file specification or merely a logical name.

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.

- 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.

DETERMINISTIC
NOT DETERMINISTIC

These clauses are synonyms for the VARIANT and NOT VARIANT clauses for conformance to the SQL/PSM standard.

The DETERMINISTIC clause indicates that the same inputs to the function will generate the same output. It is the same as the NOT VARIANT clause.

The NOT DETERMINISTIC clause indicates that the output of the function does not depend on the inputs. It is the same as the VARIANT clause.

DROP EDIT STRING

This clause removes any EDIT STRING that was previously defined for the function. No error is reported if there is no current edit string.

EDIT STRING is 'string'

Allows the value returned from the function invocation to be implicitly formatted using the EDIT STRING associated with the function. This is similar to defining an EDIT STRING on a column or domain. The EDIT STRING clause is only used by Interactive SQL.

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.

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.

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.

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

RENAME TO
Changes the name of the function being altered. See the RENAME Statement for further discussion. If the new name is the name of a synonym then an error will be raised.

RETURNS NULL ON NULL INPUT
CALLED ON NULL INPUT
These clauses control how an external function is invoked when one or more of the function arguments is NULL. The CALLED ON NULL INPUT clause specifies that the function should be executed normally. A normal execution when the PARAMETER STYLE GENERAL clause is specified means that SQL should return a run-time error when the NULL value is detected.

The RETURNS NULL ON NULL INPUT clause instructs Oracle Rdb to avoid the function call and just return a NULL result. This option is valuable for library functions such as SIN, COS, CHECKSUM, SOUNDEX, and so on, that usually return an UNKNOWN result if an argument is NULL.

The CALLED ON NULL INPUT clause is the default.

VARIANT
NOT VARIANT
These clauses are synonyms for the DETERMINISTIC and NOT DETERMINISTIC clauses for conformance to the SQL/PSM standard. The DETERMINISTIC clause indicates that the same inputs to the function will generate the same output. It is the same as the NOT VARIANT clause. The NOT DETERMINISTIC clause indicates that the output of the function does not depend on the inputs. It is the same as the VARIANT clause.

Usage Notes

- You require ALTER privilege on the specified procedure to execute this statement. If the procedure is part of a module then you must have ALTER privilege on that module.
- All of the attributes of the ALTER FUNCTION statement, with the exception of the COMPILE, COMMENT, VARIANT, DETERMINISTIC and RENAME TO clauses, apply only to external functions. An error is returned if the referenced function is an SQL stored function.
- The ALTER FUNCTION statement causes the RDB$LAST_ALTERED column of the RDB$ROUTINES table for the named function to be updated with the transaction's time stamp.
- The RENAME TO clause requires synonyms be enabled for this database. Refer to the ALTER DATABASE Statement SYNONYMS ARE ENABLED clause. Note that these synonyms may be deleted if they are no longer used by database definitions or applications.
- If a routine has many dependencies then using ALTER is preferred over a DROP . . . CASCADE and CREATE command sequence because it maintains the existing dependency information that exists between this routine and any trigger, stored routine or other database object.

Examples

Example 1: Changing a function to be NOT DETERMINISTIC

When a function is created it is assumed to be DETERMINISTIC. That is, given the same input values it should return the same result. When a routine has no parameters, such as the GET_TIME function shown below, then there is never any variation in the input. In this case the function should have been defined as NOT DETERMINISTIC to ensure that the Rdb optimizer calls it for each row processed, instead of using the previously returned result for each row.

Although DROP FUNCTION and CREATE FUNCTION could have performed the same function, ALTER FUNCTION preserves the dependencies that exist in the database.

```
SQL> alter function GET_TIME
cont>     not deterministic
cont>     comment 'Fetch time from clock'
cont>     /       'Every call must be executed, so change to be'
cont>     /       'NOT DETERMINISTIC';
SQL>
SQL> show function GET_TIME;
Information for function GET_TIME

 Function is Not Deterministic (variant)
 Function ID is: 262
 External Location is: SYS$SHARE:CLOCKSHR.EXE
 Entry Point is: GET_TIME
 Comment:       Fetch time from clock
                Every call must be executed, so change to be
                NOT DETERMINISTIC
 Language is: COBOL
 GENERAL parameter passing style used
 Number of parameters is: 0

Parameter Name                 Data Type        Domain or Type
--------------                 ---------        --------------
                               TIME(2)
        Function result datatype
        Return value is passed by value
```

Example 2: Using the DROP EDIT STRING clause

```
SQL> create function lib$lp_lines () returns integer;
cont> external language general
cont> general parameter style
cont> edit string 'S9(9)';
SQL> show function lib$lp_lines
Information for function LIB$LP_LINES

 Function ID is: 4
 Edit String:   S9(9)
 Language is: GENERAL
 GENERAL parameter passing style used
 Number of parameters is: 0

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

SQL> alter function lib$lp_lines drop edit string;
```

ALTER INDEX Statement

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

- Change the characteristics of index nodes (sorted indexes only)
- Change the names of the storage areas that contain the index
- Enable or disable logging to the .aij and .ruj files
- Alter index partitions
- Change a partition name
- Change the description of a partition
- Specify whether or not the index is UNIQUE

You cannot change:

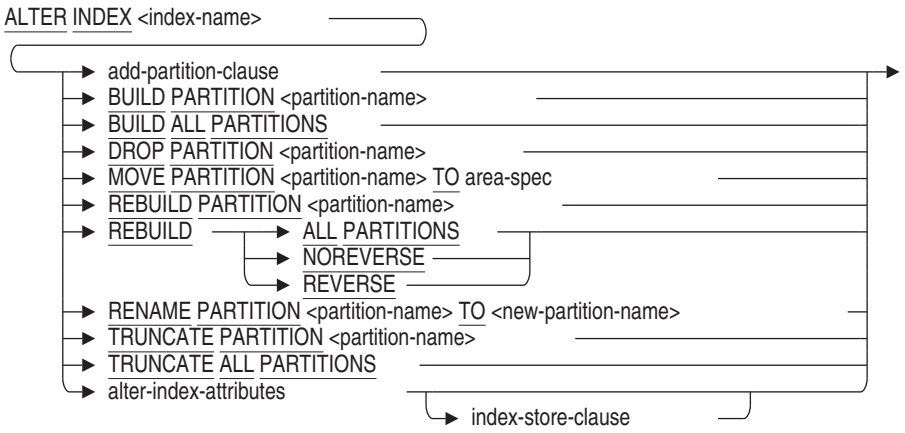
- The columns that comprise an index
- A hashed index to a sorted index
- A sorted index to a hashed index
- A sorted, nonranked index to a sorted, ranked index
- A sorted, ranked index to a sorted, nonranked index
- The duplicates compression of a sorted, ranked index

Environment

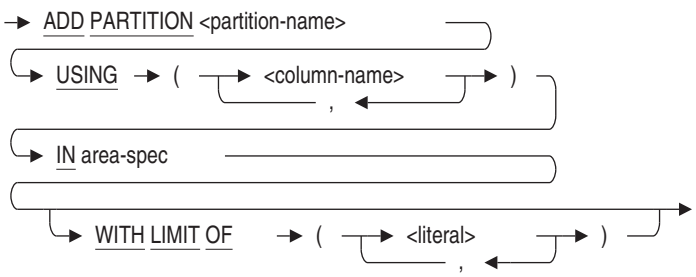
You can use the ALTER INDEX statement:

- 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

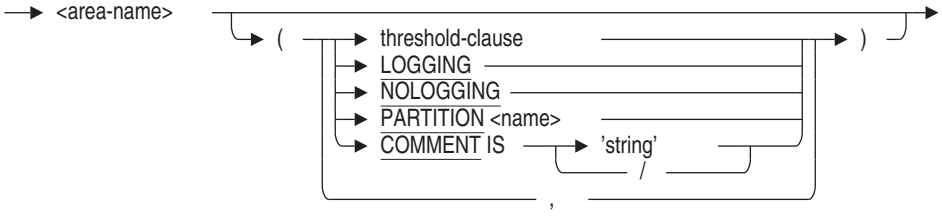
Format



add-partition-clause =

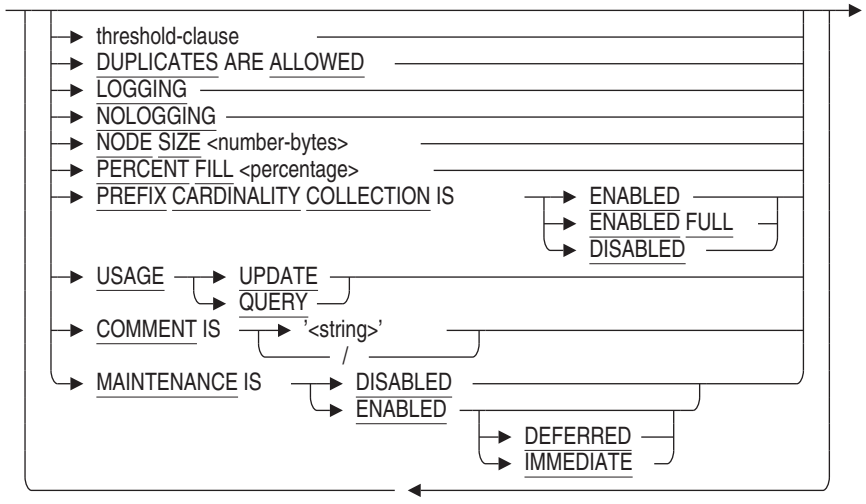


area-spec =

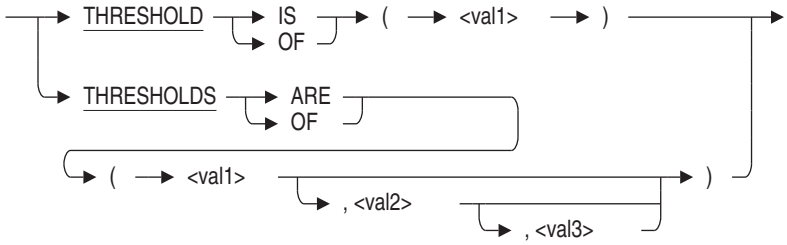


ALTER INDEX Statement

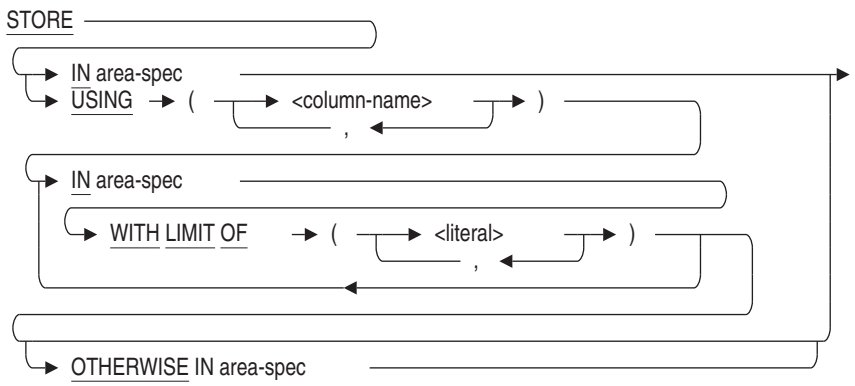
alter-index-attributes=



threshold-clause =



index-store-clause =



Arguments

ADD PARTITION partition-name

Adds the named partition to an existing hashed index. The partition name must be unique within the index being altered.

No other clauses may appear in the same ALTER INDEX statement. See the Usage Notes for additional information about using the ADD PARTITION clause.

BUILD ALL PARTITIONS

This clause operates on an index in build-pending state (created using MAINTENANCE IS ENABLED DEFERRED) and builds all incomplete partitions. If the index is not in build-pending state then the statement completes successfully with a warning.

No other clauses may appear in the same ALTER INDEX statement.

BUILD PARTITION partition-name

This clause operates on an index in build-pending state (created using MAINTENANCE IS ENABLED DEFERRED) and builds the named partition. If the index is not in build-pending state then the statement completes successfully with a warning.

No other clauses may appear in the same ALTER INDEX statement.

COMMENT IS 'string'
Adds a comment about the index. SQL displays the text of the comment when it executes a SHOW INDEX statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

DROP PARTITION partition-name
Specifies that the data in the named partition be migrated to the next partition in the map and the named partition be dropped. The last partition in the index cannot be dropped. The referenced storage area is not dropped, only the index partition stored in that area.

DUPLICATES ARE ALLOWED
Converts a UNIQUE index to a non-unique index. An index altered in this manner allows duplicate key values into the index. Note that there is no way for you to reverse this change once you commit the ALTER INDEX statement, other than by dropping and redefining the index.

IN area-spec
When specified as part of an ADD PARTITION clause, the IN area-spec inserts a new partition in the index. If you do not specify a WITH LIMIT OF clause or OTHERWISE clause, the IN area-spec clause creates a new final partition.

When specified as part of an index STORE clause, the IN area-spec clause associates the index directly with a single storage area, and all entries in the index are stored in the area you specify.

index-name
The name of the index.

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 default storage area.

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

LOGGING
NOLOGGING
The LOGGING clause specifies that updates to new index partitions should be logged in the recovery-unit journal file (.ruj) and after-image journal file (.aij).

The NOLOGGING clause specifies that updates to new index partitions should not be logged in the recovery-unit journal file (.ruj) and after-image journal file (.aij).

If no store clause is used, then these attributes provide the setting for the ALTER INDEX statement.

The LOGGING and NOLOGGING clauses are mutually exclusive; specify only one. The LOGGING clause is the default.

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 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 even when the table is in use.

Once an index has been disabled, it may be enabled again using the REBUILD PARTITION clause.

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

MAINTENANCE IS ENABLED DEFERRED
An index created using this clause does not contain index keys for the current rows in the table. Until this index is built (using ALTER INDEX . . . BUILD), the index is placed in a build-pending state. Any table with a build-pending index can not be updated using the INSERT, DELETE, or UPDATE statements.

MAINTENANCE IS ENABLED IMMEDIATE
This is the default behavior for CREATE INDEX. This clause on ALTER INDEX allows a build-pending index to be made fully operational.

MOVE PARTITION partition-name TO area-spec
Specifies that the data in the named partition be moved to the partition identified in the area-spec clause and that the current partition is to be dropped after the data is migrated. For example, this clause allows a single hashed index partition to be moved to a larger storage area when too many mixed area extends are observed.

No other clauses may appear in the same ALTER INDEX statement.

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.

This new node size is not applied to the existing index. However, it will be used in subsequent rebuild operations and by EXPORT/IMPORT to rebuild the database.

PARTITION name
Names the partition. The name can be a delimited identifier if the dialect or quoting rules are set to SQL92, SQL99 or SQL2011. Partition names must be unique within the index. If you do not specify this clause, Oracle Rdb generates a default name for the partition.

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 85 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.

PREFIX CARDINALITY COLLECTION IS DISABLED
This setting disables the cardinality collection and, instead, uses a fixed scaling algorithm which assumes a well balanced index tree. The action of this clause will also set the existing index column cardinalities to zero.

PREFIX CARDINALITY COLLECTION IS ENABLED
This is the default behavior for CREATE INDEX. The Oracle Rdb optimizer collects approximate cardinality values for the index columns to help in future query optimization. Note that no extra I/O is incurred to collect these values and, therefore, adjacent key values from other index nodes can not be checked. Hence, some inaccuracy may be seen for these indexes. In most cases, this is adequate for query optimizations. If this clause is used on an index that is currently set to PREFIX CARDINALITY COLLECTION DISABLED, the RMU Collect Optimizer_Statistics command needs to be executed as soon as possible to load the correct values.

PREFIX CARDINALITY COLLECTION IS ENABLED FULL
This setting requests that extra I/O be performed, if required, to ensure that the cardinality values reflect the key value changes of adjacent index nodes. If this clause is used on an index which is currently set to PREFIX

CARDINALITY COLLECTION DISABLED or ENABLED, the RMU Collect Optimizer_Statistics command needs to be executed as soon as possible to load the correct values.

REBUILD ALL PARTITIONS
REBUILD REVERSE
REBUILD NOREVERSE

This clause combines the TRUNCATE and BUILD actions into a single function. No other clauses may appear in the same ALTER INDEX statement.

Oracle Rdb supports the REVERSE keyword on the ALTER INDEX statement as part of the REBUILD action. This clause requests that the named index be rebuilt as REVERSE key index.

An existing SORTED or SORTED RANKED index can be converted to a REVERSE index by using this variation of the REBUILD ALL PARTITIONS clause. An already REVERSE index can be changed to a non-REVERSE index using the NOREVERSE keyword.

The following example shows an existing index being converted to a REVERSE index.

```
SQL> alter index DOCUMENTS
cont>  rebuild reverse;
SQL>
```

REBUILD PARTITION partition-name
This clause combines the TRUNCATE and BUILD actions into a single function for the named partition. No other clauses may appear in the same ALTER INDEX statement.

RENAME PARTITION partition-name TO new-partition-name
Changes the name of a partition. This clause can be applied to all types of indexes. It is particularly useful for specifying a more meaningful name for the default partition. Use the SHOW INDEX (PARTITION) statement to display the default names of the partitions. See Example 4 in the Examples section.

No other clauses may appear in the same ALTER INDEX statement.

threshold-clause
See the CREATE INDEX Statement for a description of the threshold-clause.

TRUNCATE ALL PARTITIONS
This clause operates in a similar way to TRUNCATE TABLE, but just on one index. The index is automatically set to MAINTENANCE IS ENABLED DEFERRED (i.e. build-pending state) if it was currently ENABLED IMMEDIATE. Otherwise is stays in a disabled state.

No other clauses may appear in the same ALTER INDEX statement.

TRUNCATE PARTITION partition-name
This clause operates on just the named index partition. The index is automatically set to MAINTENANCE IS ENABLED DEFERRED (that is, build-pending state) if it was currently ENABLED IMMEDIATE. Otherwise is stays in a disabled state.

No other clauses may appear in the same ALTER INDEX statement.

USAGE UPDATE
USAGE QUERY
Specifies a PERCENT FILL value appropriate for update-intensive 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.

You should specify either the PERCENT FILL or USAGE clause, but not both. If neither PERCENT FILL nor USAGE clauses are specified, then the default will be 85 percent.

You cannot specify this argument in an ALTER INDEX statement that refers to a HASHED index.

USING (column-name)
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 of 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 leading columns of the index key.

WITH LIMIT OF (literal)
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 area. For

multicolumn index keys, specify a literal value for each column listed in the USING clause.

The WITH LIMIT OF clause must specify a new unique set of values for the partition. The number of literals in the list 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.

Usage Notes

- ALTER INDEX ... REBUILD PARTITION or REBUILD ALL PARTITIONS is primarily designed for format MIXED partitions. If you execute this command on format UNIFORM partitions, then the database administrator should be aware that the affected storage area may extend. For UNIFORM storage areas, Rdb employs a *fast logical area delete* mechanism for removing the data from the index partition. While this process is fast, it has a side effect of locking those pages for the duration of the session. Therefore, the build phase of the ALTER INDEX cannot reuse those previously released pages and must allocate unused pages, possibly extending the storage area.

  Instead, Oracle recommends that the ALTER INDEX ... REBUILD be recoded to use ALTER INDEX ... TRUNCATE, a DISCONNECT ALL, and then a final ALTER INDEX ... BUILD. This is shown in the code outline below.

```
attach 'filename ...';
alter index ... truncate all partitions;
commit;
disconnect all;
attach 'filename ...';
alter index ... build all partitions;
commit;
```

- 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 complete 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 lock out attempts to query that object. These locks are held until the DDL operation is committed or rolled back.

- If an index is already defined as REVERSE then REBUILD REVERSE is equivalent to REBUILD ALL PARTITIONS.
- If an index is not defined as REVERSE then REBUILD NOREVERSE is equivalent to REBUILD ALL PARTITIONS.
- The clause REVERSE may not be applied to a HASHED index.
- When applying REVERSE to an existing index, any column defined as DESC will be modified to remove the descending ordering.
- 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.
- Note the difference between a partition and a storage area. A **partition** is a logical entity and a **storage area** is a physical entity. A table or index partition resides in a single storage area, but many such partitions from other tables and indexes can coexist in the same storage area. When a partition is said to be dropped, it is the logical entity within the storage area that is deleted. You move the data explicitly using the MOVE PARTITION clause or implicitly using the DROP PARTITION and the ADD PARTITION clauses.

  To drop the physical storage area file, use the ALTER DATABASE statement with the DROP STORAGE AREA clause.
- The partition-name that you specify must be unique within the index being altered. The name is stored in the system table RDB$STORAGE_MAP_ AREAS in the column RDB$PARTITION_NAME so that it can be used with other partition-related statements.
- Use the COMMENT ON statement to add comments to existing partitions. (See the COMMENT ON Statement for syntax.) This statement can be applied to all types of indexes.

The partition-name must exist in the index being referenced. The old comment (if one existed), is deleted and replaced with the new text you specify and becomes permanent when a COMMIT statement is executed.

- If the INDEX_STATS flag is enabled then the ALTER INDEX command logs messages to the file specified by the RDMS$DEBUG_FLAGS_OUTPUT logical name (or the standard output device, if the flag if not defined) to report the progress of the ALTER INDEX statement. The INDEX_STATS flag is enabled by doing one of the following:
  - Using the SET FLAGS 'INDEX_STATS' command.
  - Defining the RDMS$DEBUG_FLAGS logical to "Ai".
  - Defining the RDMS$SET_FLAGS logical name to INDEX_STATS.

Note

The read/write I/O statistics shown in Example 3 in the Examples section are not displayed if STATISTICS COLLECTION IS DISABLED on the database or if the logical name RDM$BIND_STATS_ENABLED is defined as 0.

- The following notes apply to the ADD, DROP, and MOVE PARTITION clauses:
  - Currently, these clauses are supported for hashed indexes only. Support for sorted indexes will be provided in a future release.
  - When you add, drop, or move a partition, there must be no active queries compiled against the table. This includes declared cursors in the current session and other applications that have referenced the table. As with other ALTER INDEX clauses, exclusive access to the table is required during the current transaction.
  - The SHOW INDEX or SHOW TABLE (INDEX) command displays the original source of the index definition with the ADD, DROP, or MOVE PARTITION source appended. See Example 3 in the Examples section. Use the RMU Extract command with the Item=Index qualifier to see the current index definition with any additional partitions merged into the CREATE INDEX syntax, dropped partitions omitted from the CREATE INDEX syntax, or updated partitions (for moved partitions) in the CREATE INDEX syntax.

- The following notes apply to the ADD PARTITION clause only:
  - The area-spec clause allows a PARTITION clause to name the partition. If you use the area-spec clause, the partition name it specifies must be the same as the partition name specified in the ADD PARTITION clause.
  - Oracle Rdb stores the partition name in the system table RDB$STORAGE_MAP_AREAS so that it can be used with other partition-related statements. The name is validated and must be unique per index.
  - To add partitions to an index, the index must have been created with a STORE clause so that additional partitions can be added.
  - The USING clause must list the same column names in the same order as in the original index definition.
  - If no WITH LIMIT OF clause is specified, then the partition is added at the end of the index as an OTHERWISE partition. If an OTHERWISE partition already exists for this index then an error is reported.
  - The WITH LIMIT OF clause must specify a new unique set of values for the partition. A literal value must exist for each column listed in the USING clause.
  - When a new final partition or an OTHERWISE partition is successfully added, no I/O to the index is required. That is, no data in the index needs to be relocated.
  - The ADD PARTITION clause reads the RDB$SYSTEM_RECORD rows that are stored on each page of a MIXED area and locates the hash buckets for the current index. Any hash keys that fall into the new partition are moved (with any associated duplicates) to the new partition. Any hash keys that do not belong in the newly added area are not moved.

Note

If this hashed index is used in a PLACEMENT VIA INDEX clause of a storage map, then those placed table rows are not moved by the ADD PARTITION clause. However, the new hashed index partition will correctly reference those rows even though they will no longer be stored adjacent to the hash bucket.

- If you attach to the database using the RESTRICTED ACCESS clause, then all partitions (and system record areas) are reserved for EXCLUSIVE access.

  These areas are also reserved for EXCLUSIVE access if the table appears in the RESERVING clause of the current transaction (either a DECLARE TRANSACTION or SET TRANSACTION statement) with an EXCLUSIVE mode.

  Otherwise, the default action is to reserve the new and subsequent partition of the index for PROTECTED WRITE. The RDB$SYSTEM_RECORD of the new partition is reserved for SHARED WRITE and the RDB$SYSTEM_RECORD of the existing partition is reserved for SHARED READ mode.

  Using EXCLUSIVE access to the partitions limits concurrent access to those storage areas by other users of the RDB$SYSTEM_RECORD, for instance, if other indexes are stored in that storage area. However, EXCLUSIVE access has the benefit of eliminating I/O to the associated snapshot files and reducing the virtual memory requirements of this operation. Therefore, Oracle Corporation recommends using EXCLUSIVE mode when possible to reduce the elapsed time of the ALTER INDEX operation. A COMMIT operation should be performed as soon as possible upon completion of the operation so that locks on the table are released.

  If the NOLOGGING clause is used (or the RDMS$CREATE_LAREA_NOLOGGING logical name is defined), then the hash buckets and duplicate nodes written to the new partition are not journaled. However, the updates to the existing RDB$SYSTEM_RECORD in that partition, and the deletions performed on the following partition, are journaled.

- The following notes apply to the DROP PARTITION clause only:
  - The partition-name must exist in the index being altered. The name is checked against the system table RDB$STORAGE_MAP_AREAS and the column RDB$PARTITION_NAME.
  - The index must have been created with a STORE clause, so that partitions can be referenced.
  - There must be no active queries compiled against this table. This includes declared cursors in the current session, or other applications that have referenced the table. As with other ALTER INDEX statements, exclusive access to the table is required during the current transaction.

- The DROP PARTITION clause reads the RDB$SYSTEM_RECORD rows that are stored on each page of a MIXED area and locates the hash buckets for the current index. All hash keys are moved (with any associated duplicates) to the next partition.

Note

If this hashed index is used in a PLACEMENT VIA INDEX clause of a storage map, then those placed table rows are not moved by the DROP PARTITION clause. However, the hashed index will still correctly reference those rows even though they will no longer be stored adjacent to the hash bucket.

- If you attach to the database using the RESTRICTED ACCESS clause, then all partitions (and system record areas) will be reserved for EXCLUSIVE access.

  These areas will also be reserved for EXCLUSIVE access if the table appears in the RESERVING clause of the current transaction (either a DECLARE TRANSACTION or SET TRANSACTION statement) with an EXCLUSIVE mode.

  Otherwise, the default action is to reserve the old partition and its RDB$SYSTEM_RECORD for EXCLUSIVE access. The following partition is reserved for PROTECTED WRITE, and its RDB$SYSTEM_RECORD is reserved for SHARED WRITE.

  Using EXCLUSIVE access to the partitions will limit concurrent access to those storage areas by other users of the RDB$SYSTEM_RECORD, for instance, if other indexes are stored in that storage area. However, EXCLUSIVE access eliminates I/O to the associated snapshot files and reduces the virtual memory requirements of this operation. Therefore, Oracle Corporation recommends using EXCLUSIVE mode when possible to reduce the elapsed time of the ALTER INDEX operation. A COMMIT operation should be performed as soon as possible upon completion of the operation so that locks on the table are released.

- The following notes apply to the MOVE PARTITION clause only:
  - The partition-name must exist in the index being altered. The name is checked against the system table RDB$STORAGE_MAP_AREAS and the column RDB$PARTITION_NAME.

- The MOVE PARTITION clause can also rename the partition by including a PARTITION clause in the area-spec clause. The partition-name specified by this clause must have the same name as specified for the MOVE PARTITION or specify a unique name not already used by this index.
- The index must have been created with a STORE clause, so that partitions can be referenced.
- The MOVE PARTITION clause reads the RDB$SYSTEM_RECORD rows which are stored on each page of a MIXED area and locates the hash buckets for the current index. All hash keys will be moved (with any associated duplicates) to the new storage area associated with this index.

Note

If this hashed index is used in a PLACEMENT VIA INDEX clause of a storage map, then those placed table rows are not moved by the MOVE PARTITION clause. However, the hashed index will still correctly reference those rows even though they will no longer be stored adjacent to the hash bucket.

- If you attach to the database using the RESTRICTED ACCESS clause, then all partitions (and system record areas) will be reserved for EXCLUSIVE access.

  These areas will also be reserved for EXCLUSIVE access if the table appears in the RESERVING clause of the current transaction (either a DECLARE TRANSACTION or SET TRANSACTION statement) with an EXCLUSIVE mode.

  Otherwise, the default action is to reserve the old partition and its RDB$SYSTEM_RECORD for EXCLUSIVE access. The following partition is reserved for PROTECTED WRITE, and its RDB$SYSTEM_RECORD is reserved for SHARED WRITE.

  Using EXCLUSIVE access to the partitions will limit concurrent access to those storage areas by other users of the RDB$SYSTEM_RECORD, for instance, if other indexes are stored in that storage area. However, EXCLUSIVE access eliminates I/O to the associated snapshot files and reduces the virtual memory requirements of this operation. Therefore, Oracle Corporation recommends using EXCLUSIVE mode when possible to reduce the elapsed time of the ALTER INDEX operation.

A COMMIT operation should be performed as soon as possible upon completion of the operation so that locks on the table are released.

- If the NOLOGGING clause is used (or if the RDMS$CREATE_LAREA_NOLOGGING logical name is defined) then the hash buckets and duplicate nodes written to the new partition are not journaled. However, the updates to the existing RDB$SYSTEM_RECORD in that partition, and the deletes performed on the following partition, are journaled.

- The following usage notes apply to the RENAME PARTITION clause only:
  - The partition-name must exist in the index being altered. The name is checked against the system table RDB$STORAGE_MAP_AREAS and the column RDB$PARTITION_NAME.
  - The new name replaces the current name and becomes permanent when a COMMIT is executed.
  - This clause can be applied to both sorted and hashed indexes.
- ALTER INDEX . . . TRUNCATE PARTITION <partition-name> is ideal for large indexes that need to be deleted. It allows parts of the index to be deleted a little at a time. When DROP INDEX is finally used, the truncated partitions will not be reprocessed.
- The TRUNCATE TABLE statement will also truncate partitions of any build-pending indexes and change the state to enabled (because the empty index is complete for an empty table). Any index that has maintenance disabled will not be processed by the TRUNCATE TABLE statement.
- Oracle Corporation recommends using SET FLAGS with the INDEX_STATS option when using any of the following ALTER INDEX clauses:
  - BUILD PARTITION and BUILD ALL PARTITIONS
  - REBUILD PARTITION and REBUILD ALL PARTITIONS
  - TRUNCATE PARTITION and TRUNCATE ALL PARTITIONS

  The traced output describes the action of these clauses.
- REBUILD PARTITION, TRUNCATE PARTITION and BUILD PARTITION all leave the index in build-pending state and an ALTER INDEX . . . MAINTENANCE IS ENABLED step must be executed after all partitions have been built. A warning is generated in interactive SQL to remind the database administrator that the index is incomplete.

```
SQL> alter index PERSON_INDEX_S
cont>     rebuild partition P3;
%RDB-W-META_WARN, metadata successfully updated with the reported warning
-RDMS-W-IDXBLDPEND, index in build pending state - maintenance is disabled
```

- The BUILD ALL and REBUILD ALL operations automatically enable maintenance on the index.
- The ALTER INDEX statement can be used to build all or part of the index on a table reserved in DATA DEFINITION mode.

  The following clauses are supported:

  – BUILD PARTITION and BUILD ALL PARTITIONS
  – REBUILD PARTITION and REBUILD ALL PARTITIONS
  – TRUNCATE PARTITION and TRUNCATE ALL PARTITIONS
  – COMMENT IS

  The BUILD and REBUILD PARTITION operators are best suited to concurrent execution as they may require I/O to construct the new index partition.

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 COLL_COLLEGE_CODE
cont> maintenance is disabled;
SQL> show index COLL_COLLEGE_CODE
Indexes on table COLLEGES:
COLL_COLLEGE_CODE               with column COLLEGE_CODE
  No Duplicates allowed
  Type is Sorted
  Key suffix compression is DISABLED
  Index is no longer maintained
  Node size  430
```

Example 2: Changing a Unique Index to Non-Unique

```
SQL> show table (index) DEPARTMENTS
Information for table DEPARTMENTS
```

```
Indexes on table DEPARTMENTS:
DEPARTMENTS_INDEX               with column DEPARTMENT_CODE
  No Duplicates allowed
  Type is Sorted
  Key suffix compression is DISABLED
  Node size  430

SQL> insert into DEPARTMENTS (DEPARTMENT_CODE) values ('SUSO');
%RDB-E-NO_DUP, index field value already exists;
duplicates not allowed for DEPARTMENTS_INDEX
SQL> alter index DEPARTMENTS_INDEX duplicates are allowed;
SQL> insert into DEPARTMENTS (DEPARTMENT_CODE) values ('SUSO');
1 row inserted
SQL> show table (index) DEPARTMENTS
Information for table DEPARTMENTS

Indexes on table DEPARTMENTS:
DEPARTMENTS_INDEX               with column DEPARTMENT_CODE
  Duplicates are allowed
  Type is Sorted
  Key suffix compression is DISABLED
  Node size  430
```

Example 3: Adding an Index Partition Before and After the Final Partition

```
SQL> CREATE UNIQUE INDEX EMPLOYEES_INDEX
cont>     ON EMPLOYEES (EMPLOYEE_ID)
cont>     TYPE IS HASHED
cont>     STORE USING (EMPLOYEE_ID)
cont>         IN JOBS WITH LIMIT OF ('00999');
SQL> COMMIT;
SQL> -- To add a partition before the final partition requires
SQL> -- that the final partition (which now follows the new partition)
SQL> -- be scanned and matching keys moved to the new partition.
SQL> SET TRANSACTION READ WRITE
cont>   RESERVING EMPLOYEES for EXCLUSIVE WRITE;
SQL> SET FLAGS INDEX_STATS;
SQL> ALTER INDEX EMPLOYEES_INDEX
cont>     ADD PARTITION NEW_EMPS_200
cont>       USING (EMPLOYEE_ID)
cont>       IN EMP_INFO WITH LIMIT OF ('00200');
~Ai alter index "EMPLOYEES_INDEX" (hashed=1, ordered=0)
~Ai add partition "NEW_EMPS_200" : area "EMP_INFO"
~Ai storage area "EMP_INFO" larea=85
~Ai splitting partition #1
~Ai split complete: total 100 keys, moved 37 (dups 0)
~Ai reads: async 136 synch 30, writes: async 57 synch 0
SQL> COMMIT;
SQL> -- Now add a partition after the final partition of
SQL> -- the index. This requires no I/O to the partition because
SQL> -- there is no following partition and therefore no keys
SQL> -- to be moved.
SQL> SET TRANSACTION READ WRITE
cont>   RESERVING EMPLOYEES FOR EXCLUSIVE WRITE;
SQL> ALTER INDEX EMPLOYEES_INDEX
cont>     ADD PARTITION NEW_EMPS_1400
cont>       USING (EMPLOYEE_ID)
cont>       IN EMPIDS_OVER WITH LIMIT OF ('01400');
~Ai alter index "EMPLOYEES_INDEX" (hashed=1, ordered=0)
~Ai add partition "NEW_EMPS_1400" : area "EMPIDS_OVER"
~Ai storage area "EMPIDS_OVER" larea=122
~Ai adding new final partition 3
SQL> COMMIT;
SQL> -- Show the index.  It shows the ADD PARTITION syntax appended
SQL> -- to the original source of the index.
SQL> SHOW INDEX EMPLOYEES_INDEX
Indexes on table EMPLOYEES:
EMPLOYEES_INDEX                 with column EMPLOYEE_ID
  No Duplicates allowed
  Type is Hashed Scattered
  Key Suffix Compression is DISABLED
Store clause:           STORE using (EMPLOYEE_ID)
                             in JOBS with limit of ('00999')
                        Add Partition partition NEW_EMPS_200
                        using (EMPLOYEE_ID)
```

```
                        in EMP_INFO with limit of ('00200')
                        Add Partition partition NEW_EMPS_1400
                        using (EMPLOYEE_ID)
                        in EMPIDS_OVER with limit of ('01400')
```

Example 4: Renaming a Partition

```
$ rmu /extract /item=index mf_personnel.rdb
.
.
.
create unique index EMPLOYEES_HASH
    on EMPLOYEES (
    EMPLOYEE_ID)
    type is HASHED
    store
        using (EMPLOYEE_ID)
            in EMPIDS_LOW(
                partition "SYS_P00076"
                )
                with limit of ('00200')
            in EMPIDS_MID(
                partition "SYS_P00077"
                )
                with limit of ('00400')
            otherwise in EMPIDS_OVER(
                partition "SYS_P00078"
                );
commit work;
$SQL$
SQL> ATTACH FILENAME MF_PERSONNEL.RDB;
SQL> ALTER INDEX EMPLOYEES_HASH
cont> RENAME PARTITION SYS_P00076 TO IDS_LOW;
SQL> ALTER INDEX EMPLOYEES_HASH
cont> RENAME PARTITION SYS_P00077 TO IDS_MID;
SQL> ALTER INDEX EMPLOYEES_HASH
cont> RENAME PARTITION SYS_P00078 TO IDS_HIGH;
SQL> COMMIT;
SQL> SHOW INDEX EMPLOYEES_HASH;
Indexes on table EMPLOYEES:
EMPLOYEES_HASH                  with column EMPLOYEE_ID
  No Duplicates allowed
  Type is Hashed Scattered
  Key Suffix Compression is DISABLED
Store clause:           STORE USING (EMPLOYEE_ID)
                             IN EMPIDS_LOW WITH LIMIT OF ('00200')
                             IN EMPIDS_MID WITH LIMIT OF ('00400')
                             OTHERWISE IN EMPIDS_OVER
                        Rename PARTITION SYS_P00076 TO IDS_LOW
                        Rename PARTITION SYS_P00077 TO IDS_MID
                        Rename PARTITION SYS_P00078 TO IDS_HIGH
```

Example 5: Creating a Large Index Partitioned Across Many Storage Areas

First, create the database definition:

```
SQL> CREATE INDEX ... MAINTENANCE IS ENABLED DEFERRED ...;
```

Next submit batch jobs to build each partition in parallel. For example, each batch job would execute a script similar to the following:

```
ATTACH 'filename testdatabase';
SET FLAGS 'index_stats';
ALTER INDEX TRANSACTIONS_INDEX BUILD PARTITION PART_1;
COMMIT;
```

Finally, after the batch jobs have completed, the database administrator must make the index active for query usage by changing the maintenance mode to ENABLED IMMEDIATE. A BUILD ALL PARTITIONS clause could be added in case any step failed (possibly due to resource limitations or a failed node).

```
SQL> SET FLAGS 'index_stats';
SQL> SET TRANSLATION READ WRITE RESERVING...FOR EXCLUSIVE WRITES;
SQL> ALTER INDEX ... BUILD ALL PARTITIONS;
SQL> ALTER INDEX ... MAINTENANCE IS ENABLED IMMEDIATE;
SQL> COMMIT;
```

This scheme has several advantages over issuing a CREATE INDEX statement directly:

- The build actions can be run in parallel, which allows better resource usage (read and sort fewer rows), and reduced execution time for the index creation.
- The partitions being processed are relatively small when compared to the full index and, therefore, smaller quantities of data will be processed. This will result in smaller .ruj files and less AIJ file space for these transactions.
- Each build partition runs in a separate transaction, can easily be repeated if a step fails, and does not require repeating the entire CREATE INDEX statement.
- If any steps have failed, they will also be repeated by the BUILD ALL PARTITIONS clause included in the script.

Example 6: Deleting a Large Index Partitioned Across Many Storage Areas

First, disable the index:

```
SQL> ALTER INDEX TRANSACTIONS_INDEX MAINTENANCE IS DISABLED;
```

Next, submit batch jobs to truncate the partitions in parallel:

```
SQL> ALTER INDEX TRANSACTIONS_INDEX TRUNCATE PARTITION PART_1;
SQL> COMMIT;
```

Finally, after the batch jobs are complete, remove the metadata:

```
SQL> DROP INDEX TRANSACTIONS_INDEX;
```

This scheme has several advantages over a issuing a DROP INDEX statement directly:

- The truncate actions can be run in parallel, which allows better resource usage and reduced execution time for the index deletion.
- The partitions being processed are relatively small when compared to the full index and, therefore, smaller quantities of data will be processed. This will result in smaller .ruj files and less AIJ file space for these transactions.
- Each truncate partition runs in a separate transaction, can easily be repeated if a step fails, and does not require repeating the entire action.
- If any steps have failed, they will also be repeated by a DROP INDEX statement.

Example 7: Using the TRUNCATE PARTITION statement

The following example illustrates using the TRUNCATE PARTITION statement for the MF_PERSONNEL database.

```
SQL> show index (partition) EMPLOYEES_HASH
Indexes on table EMPLOYEES:
EMPLOYEES_HASH                  with column EMPLOYEE_ID
  No Duplicates allowed
  Type is Hashed Scattered
  Key suffix compression is DISABLED

 Partition information for index:
  Partition: (1) SYS_P00076
   Storage Area: EMPIDS_LOW
  Partition: (2) SYS_P00077
   Storage Area: EMPIDS_MID
  Partition: (3) SYS_P00078
   Storage Area: EMPIDS_OVER

SQL> alter index employees_hash truncate partition SYS_P00077;
%RDB-W-META_WARN, metadata successfully updated with the reported warning
-RDMS-W-IDXBLDPEND, index in build pending state - maintenance is disabled
SQL> insert into employees default values;
%RDB-E-READ_ONLY_REL, relation EMPLOYEES was reserved for read access; updates
not allowed
-RDMS-F-BUILDPENDING, index in build pending state - operation not permitted
```

Until the index is made complete it will not be used by the query optimizer, nor can the table on which it is defined be updated. The SHOW INDEX command reports this state.

```
SQL> show index employees_hash
Indexes on table EMPLOYEES:
EMPLOYEES_HASH                  with column EMPLOYEE_ID
  No Duplicates allowed
  Type is Hashed Scattered
  Key suffix compression is DISABLED
  Maintenance is Deferred - build pending
```

ALTER MODULE Statement

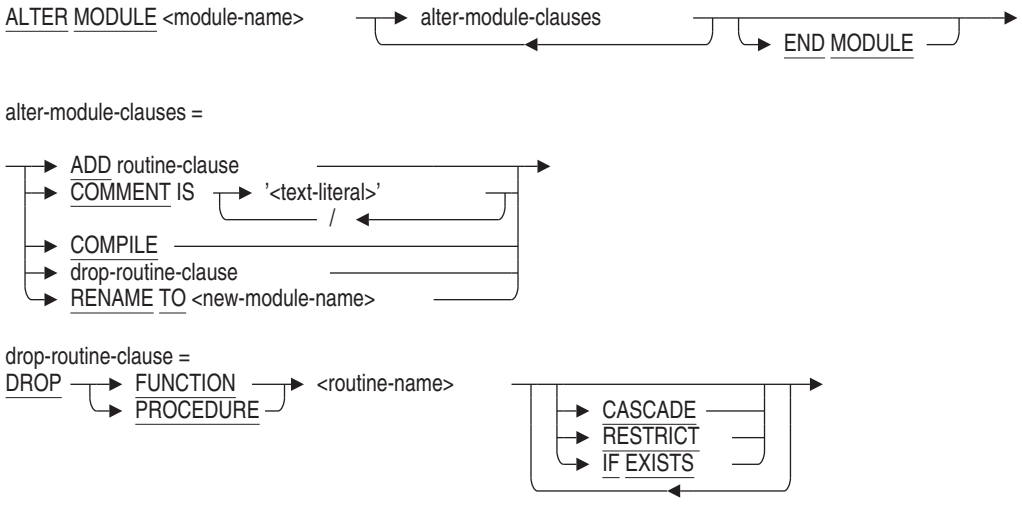
Alters a module to add or drop routines, change a comment, or compile stored routines.

Environment

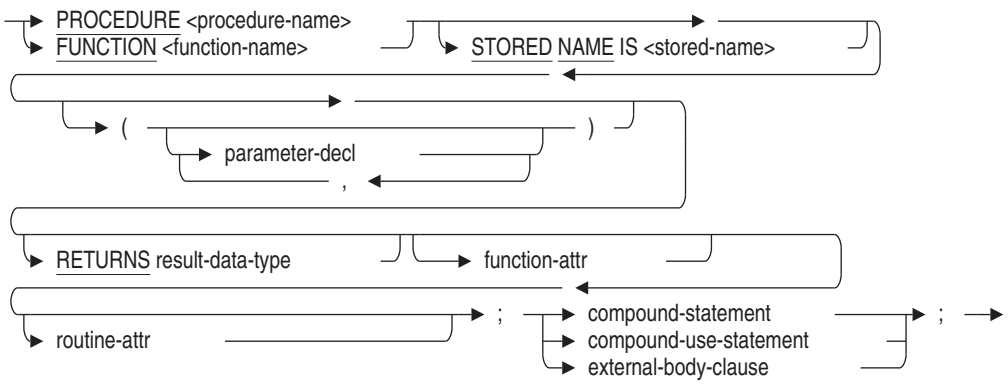
You can use the ALTER MODULE statement:

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

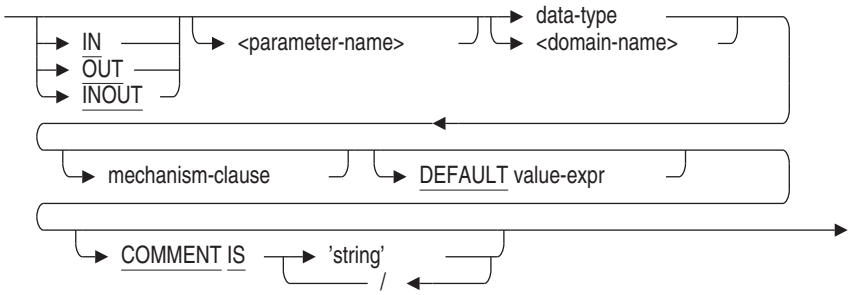
Format



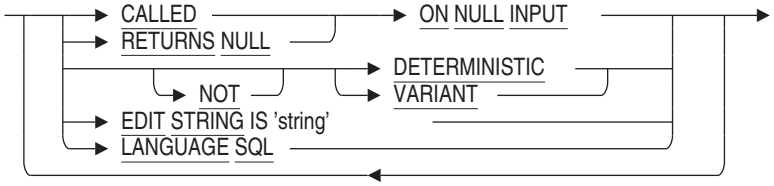
routine-clause =



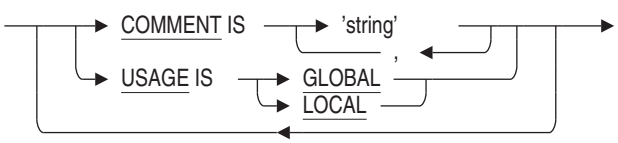
parameter-decl =



function-attr =



routine-attr =



Arguments

ADD routine-clause
Allows new functions and procedures to be added to the module. Refer to CREATE MODULE Statement for details on the routine-clause. The END MODULE clause must be used to end the ALTER MODULE clause to provide an unambiguous statement termination.

COMMENT IS 'string'
Adds a comment about the module. Enclose the comment within single quotation marks (') and separate multiple lines in a comment with a slash mark (/). This clause is equivalent to the COMMENT ON MODULE statement.

COMPILE
Recompiles stored routines in the module. Any that were marked invalid will have this flag cleared if the compile was successful.

drop-routine-clause
The DROP FUNCTION and DROP PROCEDURE clauses will drop the named routines from this module. All DROP clauses are executed prior to the COMPILE and ADD clauses in this ALTER statement.

END MODULE
This terminating clause is required when using ADD FUNCTION or ADD PROCEDURE since there is no way to distinguish between the end of a compound statement and the end of the ALTER MODULE statement.

RENAME TO
Changes the name of the module being altered. See the RENAME Statement for further discussion. If the new name is the name of a synonym then an error will be raised.

The RENAME TO clause requires synonyms be enabled for this database. Refer to the ALTER DATABASE Statement SYNONYMS ARE ENABLED clause. Note that these synonyms may be deleted if they are no longer used by database definitions or applications.

The old name will be used to create a synonym for the new name of this module. This synonym can be dropped if the name is no longer used by applications.

Usage Notes

- You require ALTER privilege on the referenced module.
- The ALTER MODULE statement causes the RDB$LAST_ALTERED column of the RDB$MODULES table for the named module to be updated with the transaction's timestamp.
- The COMPILE option forces the Rdb server to recompile all the stored (SQL) routines within the named module. External routines in the module are not affected.
- Use COMPILE when a routine has been made INVALID by the execution of a DROP ... CASCADE operation. This mechanism should be preferred over the SET FLAGS 'VALIDATE_ROUTINE' method available in previous versions.
- The ALTER MODULE statement supports the the EXTERNAL DEFAULTS created with the module. Therefore, any external routine added to the module with the ADD FUNCTION or ADD PROCEDURE clauses will inherit any missing attributes from the original module definitions.

Examples

Example 1: Changing the comment on a module

A comment can be added or changed on a module using the COMMENT IS clause as shown in this example.

```
SQL> alter module EMPLOYEES_MAINTENANCE
cont>     comment is
cont>           'routines to add and remove employee rows'
cont>     /     'Fix: also record the employees birthday';
SQL>
SQL> show module EMPLOYEES_MAINTENANCE;
Information for module EMPLOYEES_MAINTENANCE

 Header:
 EMPLOYEES_MAINTENANCE
 Comment:       routines to add and remove employee rows
                Fix: also record the employees birthday
 Module ID is: 7
```

```
 Routines in module EMPLOYEES_MAINTENANCE:
     ADD_EMPLOYEE
     IS_CURRENT_EMPLOYEE
     REMOVE_EMPLOYEE
```

Example 2: Revalidating all routines in a module

The COMPILE clause can be used to check each stored procedure or function to ensure that it can be executed. If the compile fails it will report the first reason, in this example a missing table.

```
SQL> alter module EMPLOYEES_MAINTENANCE compile;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-OBSOLETE_METADA, request references metadata objects that no longer exist
-RDMS-F-BAD_SYM, unknown relation symbol - ARCHIVE_EMPLOYEES
```

Example 3: Replacing a routine in a module

The following example creates a simple module and shows the effect of DROP TABLE . . . CASCADE. That is, the procedure REMOVE_EMPLOYEE is marked as invalid. The COMPILE clause is used to attempt to re-validate the procedure, however, a referenced table no longer exists. After replacing the table the COMPILE completes successfully.

```
SQL> set dialect 'sql99';
SQL> attach 'file PERSONNEL1';
SQL>
SQL> create table EMPLOYEES
cont>     (employee_id       integer,
cont>      last_name         char(40),
cont>      first_name        char(40),
cont>      birthday          date,
cont>      start_date        date default current_date);
SQL>
SQL> create table ARCHIVE_EMPLOYEES
cont>     (employee_id       integer,
cont>      last_name         char(40),
cont>      first_name        char(40),
cont>      archive_date      date default current_date);
SQL>
SQL> create module EMPLOYEES_MAINTENANCE
cont>
cont>     procedure REMOVE_EMPLOYEE (in :employee_id integer);
cont>     begin
cont>     -- take copy of the old row
cont>     insert into ARCHIVE_EMPLOYEES
cont>         (employee_id, last_name, first_name)
cont>         select employee_id, last_name, first_name
cont>         from EMPLOYEES
cont>         where employee_id = :employee_id;
```

```
cont>     -- remove the old row
cont>     delete from EMPLOYEES
cont>         where employee_id = :employee_id;
cont>     end;
cont>
cont>     procedure ADD_EMPLOYEE
cont>         (in :employee_id integer,
cont>          in :last_name char(40),
cont>          in :first_name char(40),
cont>          in :birthday date);
cont>     insert into EMPLOYEES
cont>         (employee_id, last_name, first_name, birthday)
cont>         values (:employee_id, :last_name, :first_name, :birthday);
cont>
cont> end module;
SQL>
SQL> show module EMPLOYEES_MAINTENANCE
Information for module EMPLOYEES_MAINTENANCE

 Header:
 EMPLOYEES_MAINTENANCE
 Module ID is: 7

 Routines in module EMPLOYEES_MAINTENANCE:
     ADD_EMPLOYEE
     REMOVE_EMPLOYEE

SQL>
SQL> drop table ARCHIVE_EMPLOYEES cascade;
SQL>
SQL> show procedure REMOVE_EMPLOYEE;
Information for procedure REMOVE_EMPLOYEE

Current state is INVALID
        Can be revalidated
 Procedure ID is: 8
 Source:
 REMOVE_EMPLOYEE (in :employee_id integer);
    begin
    -- take copy of the old row
    insert into ARCHIVE_EMPLOYEES
        (employee_id, last_name, first_name)
        select employee_id, last_name, first_name
        from EMPLOYEES
        where employee_id = :employee_id;
    -- remove the old row
    delete from EMPLOYEES
        where employee_id = :employee_id;
    end
 No description found
 Module name is: EMPLOYEES_MAINTENANCE
 Module ID is: 7
 Number of parameters is: 1
```

```
Parameter Name                  Data Type        Domain or Type
--------------                  ---------        --------------
EMPLOYEE_ID                     INTEGER
        Parameter position is 1
        Parameter is IN (read)
        Parameter is passed by reference

SQL>
SQL> -- COMPILE reports the missing table
SQL> alter module EMPLOYEES_MAINTENANCE compile;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-OBSOLETE_METADA, request references metadata objects that no longer exist
-RDMS-F-BAD_SYM, unknown relation symbol - ARCHIVE_EMPLOYEES
SQL>
SQL> create table ARCHIVE_EMPLOYEES
cont>     (employee_id        integer,
cont>      last_name          char(40),
cont>      first_name         char(40),
cont>      birthday           date,
cont>      archive_date       date default current_date);
SQL>
SQL> -- new table definition is compatible
SQL> alter module EMPLOYEES_MAINTENANCE compile;
SQL>
SQL> alter module EMPLOYEES_MAINTENANCE
cont>     comment is
cont>           'routines to add and remove employee rows'
cont>     /     'Fix: also record the employees birthday'
cont>
cont>     drop procedure REMOVE_EMPLOYEE if exists
cont>
cont>     add procedure REMOVE_EMPLOYEE (in :employee_id integer);
cont>     begin
cont>     -- take copy of the old row
cont>     insert into ARCHIVE_EMPLOYEES
cont>         (employee_id, last_name, first_name, birthday)
cont>         select employee_id, last_name, first_name, birthday
cont>         from EMPLOYEES
cont>         where employee_id = :employee_id;
cont>     -- remove the old row
cont>     delete from EMPLOYEES
cont>         where employee_id = :employee_id;
cont>     end;
cont> end module;
SQL>
SQL> show module EMPLOYEES_MAINTENANCE;
Information for module EMPLOYEES_MAINTENANCE
```

```
Header:
EMPLOYEES_MAINTENANCE
Comment:       routines to add and remove employee rows
               Fix: also record the employees birthday
Module ID is: 7

Routines in module EMPLOYEES_MAINTENANCE:
    ADD_EMPLOYEE
    REMOVE_EMPLOYEE
```

Example 4: Adding a new function to a module

In the following example the ADD clause is used to add a new function to an existing module.

```
SQL> alter module EMPLOYEES_MAINTENANCE
cont>     add function IS_CURRENT_EMPLOYEE (in :employee_id integer)
cont>         returns integer;
cont>     return (case
cont>             when exists (select *
cont>                          from EMPLOYEES
cont>                          where employee_id = :employee_id)
cont>             then 1
cont>             else 0
cont>             end);
cont> end module;
SQL>
SQL> show module EMPLOYEES_MAINTENANCE;
Information for module EMPLOYEES_MAINTENANCE

 Header:
 EMPLOYEES_MAINTENANCE
 Comment:       routines to add and remove employee rows
                Fix: also record the employees birthday
 Module ID is: 7

 Routines in module EMPLOYEES_MAINTENANCE:
     ADD_EMPLOYEE
     IS_CURRENT_EMPLOYEE
     REMOVE_EMPLOYEE
```

Example 5: ALTER MODULE statement supporting EXTERNAL DEFAULTS

```
SQL> create module DCL_SYMBOLS
cont>     external defaults (
cont>         location 'SYS$SHARE:LIBRTL.EXE'
cont>         language GENERAL
cont>         parameter style GENERAL
cont>     )
cont>
cont>   procedure LIB$SET_SYMBOL (in :symbol varchar(40) by descriptor,
cont>                             in :value_string varchar(40) by descriptor,
cont>                             in :table_type_indicator integer = 2);
cont>     external;
cont>
cont>   procedure LIB$GET_SYMBOL (in :symbol varchar(40) by descriptor,
cont>                            out :resultant_string varchar(40) by descriptor,
cont>                              out :resultant_length smallint,
cont>                              in :table_type_indicator integer = 2);
cont>     external;
cont>
cont> end module;
```

At a later time, ALTER MODULE can be executed to add other routines in the same shareable image.

```
SQL> ! Add new routine using ALTER DATABASE
SQL> !
SQL> alter module DCL_SYMBOLS
cont>   add
cont>   procedure LIB$DELETE_SYMBOL (in :symbol varchar(40) by descriptor,
cont>                                in :table_type_indicator integer = 2);
cont>     external;
cont> end module;
```

ALTER OUTLINE Statement

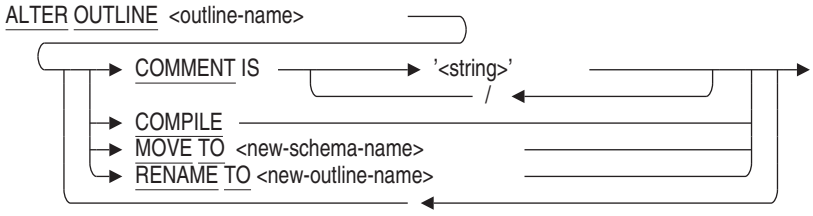
Alters an outline definition.

Environment

You can use the ALTER OUTLINE statement:

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

Format



Arguments

COMMENT IS string

The COMMENT IS clause can be used to modify the comment stored with the query outline. The COMMENT ON statement is identical in function to the ALTER OUTLINE . . . COMMENT IS clause.

This clause is equivalent to the COMMENT ON procedure.

COMPILE

The COMPILE option can be applied to query outlines that have been made invalid by DROP TABLE or DROP INDEX. If the tables and indices have been recreated then the query outline will be made valid again. For example, once re-validated the optimizer will try to use that query outline.

Note

There is a possibility that the query outline although marked valid will not be used because of changes in the index definition. There is too little information stored with the query outline to perform a complete consistency check. If possible, queries using this outline should be run to verify correct index and table usage.

If the query outline is currently valid then this clause is ignored by Oracle Rdb.

MOVE TO

MOVE TO is valid only for multischema databases. You must be attached explicitly or implicitly with the MULTISCHEMA IS ON clause. The MOVE TO clause can be used to move the query outline to a different catalog and schema. An error will be raised if this clause is specified in a non-multischema environment.

The target catalog and schema must exist in this database.

RENAME TO

The RENAME TO clause can be used to change the name of the outline. The new name must not already exist in the database.

If RENAME TO is used in a multischema database, attached with MULTISCHEMA IS ON, then only the multischema name is modified not the STORED NAME of the object. To change the STORED NAME of the query outline you must attach to the database explicitly with the MULTISCHEMA IS OFF clause (see the example below). Please note that the STORED NAME for the query outline may have been generated by Oracle Rdb.

Note

Any queries using the OPTIMIZE USING clause will also need to be changed to reference this new outline name.

Usage Notes

- You require ALTER privilege on the database to execute this statement.
- The outline name can be prefixed with a database alias name. For example,

```
SQL> attach 'ALIAS db1 FILENAME mschema_db';
SQL> alter outline db1.SHOW_TABLES_QUERY
cont>     comment is 'used to select SHOW_TAB_INDEX_01';
```

  In a multischema database the name can also include a schema name and catalog name.

Examples

Example 1: Changing the comment on a query outline

```
SQL> alter outline show_tables
cont>     comment is  'show the tables query'
cont>     /           'derived from a stored procedure';
SQL> show outline show_tables
     SHOW_TABLES
 Comment:       show the tables query
                derived from a stored procedure
 Source:
-- Rdb Generated Outline :  8-FEB-2002 16:17
create outline SHOW_TABLES
id '4D5B5CC5B46C6DD21B0E1999C0EB8BF3'
mode 0
as (
  query (
-- For loop
    subquery (
      RDB$RELATIONS 0         access path index       RDB$REL_REL_NAME_NDX
      )
    )
  )
compliance optional     ;
```

Example 2: Using the alternate COMMENT ON syntax to change the comment

```
SQL> comment on outline show_tables
cont>     is  'show the tables query'
cont>     /   'derived from the stored procedure'
cont>     /   'SHOW_TABLES';
```

Example 3: Changing the name of a query outline

```
SQL> alter outline show_tables
cont>     rename to show_the_tables;
SQL> show outline show_the_tables
     SHOW_THE_TABLES
 Comment:       show the tables query
                derived from the stored procedure
                testing new COMMENT ON OUTLINE
 Source:
-- Rdb Generated Outline :  8-FEB-2002 16:17
create outline SHOW_THE_TABLES
id '4D5B5CC5B46C6DD21B0E1999C0EB8BF3'
mode 0
as (
  query (
-- For loop
    subquery (
      RDB$RELATIONS 0         access path index       RDB$REL_REL_NAME_NDX
      )
    )
  )
compliance optional     ;
```

Example 4: This example shows setting a query outline valid after a DROP INDEX

First, our stored procedure is executed with the STRATEGY flag defined so we can see that it is using a query outline named MY_OUTLINE.

```
SQL> set flags 'strategy';
SQL> call my_procedure();
~S: Outline "MY_OUTLINE" used
Aggregate       Conjunct        Index only retrieval of relation MY_TABLE
  Index name  MY_INDEX [1:1]
```

Now the index that was used by the query (and referenced by the query outline) is dropped. This causes the query outline to be set invalid (as shown by using the WARN_INVALID flag). The query now uses sequential access strategy when the stored procedure is executed.

```
SQL> set flags 'warn_invalid';
SQL> drop index my_index;
~Xw: Outline "MY_OUTLINE" marked invalid (index "MY_INDEX" dropped)
SQL>
SQL> set flags 'strategy';
SQL> call my_procedure();
~S: Outline "MY_OUTLINE" is invalid and can not be used
Aggregate       Conjunct        Get
Retrieval sequentially of relation MY_TABLE
SQL> show outline my_outline
     MY_OUTLINE
    Outline has been marked invalid
   .
   .
   .
```

The ALTER OUTLINE ... COMPILE clause is now used to make the outline valid. The first attempt reports that the index is missing. After the index is recreated the COMPILE succeeds. Calling the stored procedure now uses this query outline.

```
SQL> alter outline my_outline compile;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-INDNOTEXI, index MY_INDEX does not exist in this database
SQL> -- must redefine the index
SQL> create index my_index on my_table (b desc);
SQL> alter outline my_outline compile;
SQL> call my_procedure();
~S: Outline "MY_OUTLINE" used
Aggregate       Conjunct        Index only retrieval of relation MY_TABLE
  Index name  MY_INDEX [1:1]
SQL>
```

Example 5: Changing the STORED NAME of a query outline in a multischema database

This example shows how to change the STORED NAME of a multischema outline. Here we explicitly provide the STORED NAME, however, the same technique can be used when SQL generates a unique STORED NAME for the outline.

```
SQL> attach 'filename mschema';
SQL> create outline SHOW_TABLE
cont>   stored name SHOW_TABLE_01
cont>   on procedure name SHOW_TABLES;
SQL> commit;
SQL> disconnect all;
SQL> attach 'filename mschema MULTISCHEMA IS OFF';
SQL> alter outline SHOW_TABLE_01
cont>    rename to SHOW_THE_TABLES;
SQL> commit;
```

ALTER PROCEDURE Statement

Allows attributes to be changed for a procedure that was created using the CREATE MODULE statement or the CREATE PROCEDURE statement.

It can be used to:

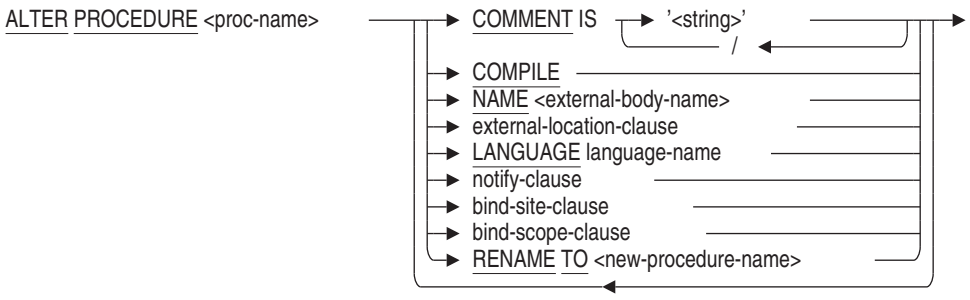
- Force a stored (SQL) procedure to be compiled (COMPILE option)
- Modify attributes of an external procedure
- Change the comment on a procedure

Environment

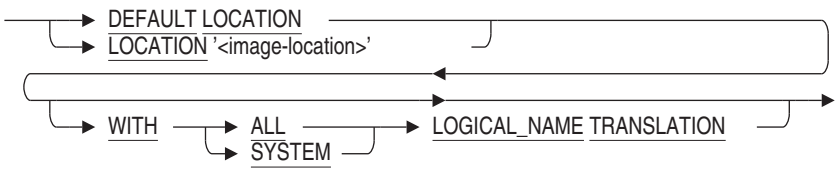
You can use the ALTER PROCEDURE statement:

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

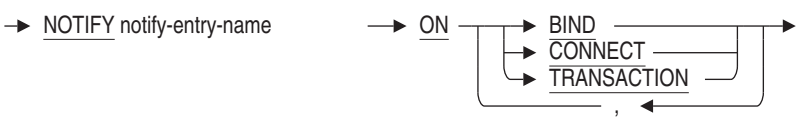
Format



external-location-clause =

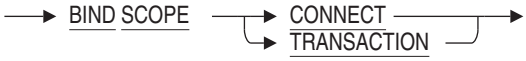


notify-clause =



bind-site-clause =

bind-scope-clause =



Arguments

BIND ON CLIENT SITE
BIND ON SERVER SITE
Selects the execution model and environment for external routine execution.

CLIENT site binding causes the external routine to be activated and executed in the OpenVMS database client (application) process. This is the default binding. This binding offers the most efficient execution characteristics, allows sharing resources such as I/O devices, and allows debugging of external routines as if they were part of the client application. However, this binding may suffer from address space limitations. Because it shares virtual memory with the database buffers, this binding is restricted to the client process system user environment, and prohibits external routine execution in cases of an application running with elevated privileges.

SERVER site binding causes the external routine to be activated in a separate process from the database client and server. The process is started on the same node at the database process. This binding offers reasonable execution characteristics, a larger address space, a true session user environment, and has no restrictions regarding client process elevated privileges. However, this binding does not permit sharing resources such as I/O devices with the client (in particular, there is no connection to the client interactive terminal), and debugging of routines is generally not possible.

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.

COMMENT IS string
Adds a comment about the procedure. SQL displays the text of the comment when it executes a SHOW PROCEDURES statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

This clause is equivalent to the COMMENT ON PROCEDURE statement.

COMPILE
The COMPILE option forces the Oracle Rdb server to recompile the stored (SQL) procedure. External procedures are not affected.

Use COMPILE when a procedure has been made invalid by the execution of a DROP . . . CASCADE operation. This mechanism is preferred over the SET FLAGS 'VALIDATE_ROUTINE' method available in previous versions.

DEFAULT LOCATION
LOCATION 'image-location'
A default or specific location for the external routine image. The resulting file specification must include the type *.exe*.

This can be an image file specification or merely a logical name.

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.

- 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.

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.

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.

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.

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

RENAME TO
Changes the name of the procedure being altered. See the RENAME Statement for further discussion. If the new name is the name of a synonym then an error will be raised.

Usage Notes

- You require ALTER privilege on the specified procedure to execute this statement. If the procedure is part of a module then you must have ALTER privilege on that module.
- All of the attributes of the ALTER PROCEDURE statement, with the exception of the COMPILE, COMMENT, and RENAME TO clauses apply only to external procedures.
- The ALTER PROCEDURE statement causes the RDB$LAST_ALTERED column of the RDB$ROUTINES table for the named procedure to be updated with the transactions time stamp.
- The RENAME TO clause requires synonyms be enabled for this database. Refer to the ALTER DATABASE Statement SYNONYMS ARE ENABLED clause. Note that these synonyms may be deleted if they are no longer used by database definitions or applications.
- If a routine has many dependencies then using ALTER is preferred over a DROP . . . CASCADE and CREATE command sequence because it maintains the existing dependency information that exists between this routine and any trigger, stored routine or other database object.

Examples

Example 1: Using ALTER PROCEDURE to target a new routine and sharable image

This example shows ALTER PROCEDURE updating the location, routine name and language for an external procedure.

```
SQL> show procedure SEND_MAIL
Information for procedure SEND_MAIL

 Procedure ID is: 261
 External Location is: SYS$SHARE:SENDMAILSHR.EXE
 Entry Point is: SEND_MAIL
 Language is: COBOL
 GENERAL parameter passing style used
 Number of parameters is: 2

Parameter Name                  Data Type        Domain or Type
--------------                  ---------        --------------
USR                             CHAR(30)
        Parameter position is 1
        Parameter is IN (read)
        Parameter is passed by reference

TXT                             VARCHAR(1000)
        Parameter position is 2
        Parameter is IN (read)
        Parameter is passed by reference

SQL> /*
***> The routine has been rewritten.  Use ALTER PROCEDURE
***> to retarget the external routine to use the new
***> implementation, instead of using DROP/CREATE
***> */
SQL>
SQL> set quoting rules 'SQL99';
SQL>
SQL> alter procedure SEND_MAIL
cont>     name "send_mail_ext"
cont>     location 'SYS$SHARE:SENDMAILSHR30.EXE'
cont>     language C
cont>     comment 'Use new V3.0 interface routine';
SQL>
```

ALTER PROFILE Statement

Alters a profile definition.

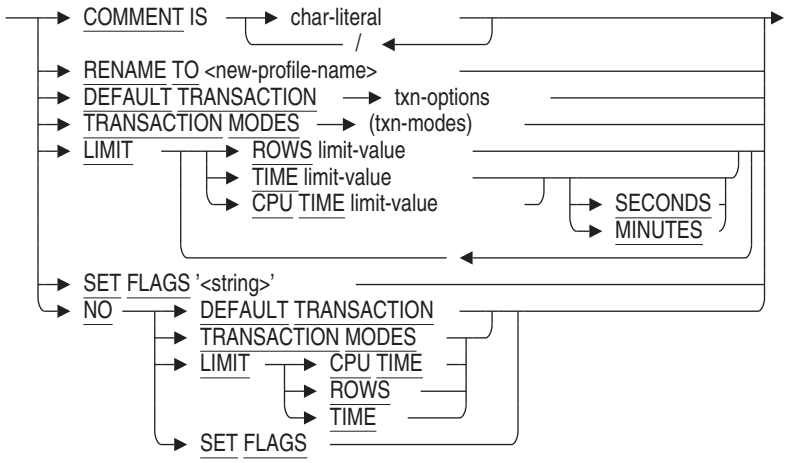
Environment

You can use the ALTER PROFILE statement:

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

Format

profile-options =



Arguments

ALIAS aliasname
When attached to multiple databases, the aliasname is required to direct the ALTER command to the appropriate database.

COMMENT IS 'string'
Adds a comment about the profile. SQL displays the text of the comment when it executes a SHOW PROFILES statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

DEFAULT PROFILE
Alters the special profile RDB$DEFAULT_PROFILE. This profile will be used by any user who is not assigned a profile using the PROFILE clause of CREATE or ALTER USER.

DEFAULT TRANSACTION
DEFAULT TRANSACTION provides a default transaction for the user. By default, Oracle Rdb starts a READ WRITE transaction if none is explicitly started. Use the DECLARE DEFAULT TRANSACTION or START DEFAULT TRANSACTION statement to make use of this definition. You can override this clause with a DECLARE or SET TRANSACTION statement.

Note

Oracle Rdb does not permit the RESERVING or EVALUATING clauses to appear in the default transaction.

LIMIT CPU TIME
NO LIMIT CPU TIME
LIMIT CPU TIME sets the maximum CPU time that can be used by the query compiler. The keyword DEFAULT indicates that no value is defined by this profile and is equivalent to NO LIMIT CPU TIME.

If a numeric value or the keyword UNLIMITED is specified then this value will be used even when the SET QUERY LIMIT CPU TIME statement is present in the session, or when the logical name RDMS$BIND_QG_CPU_TIMEOUT is defined.

NO LIMIT CPU TIME is the default. Units can be specified as seconds or minutes.

LIMIT ROWS
NO LIMIT ROWS
LIMIT ROWS sets the maximum number of rows that can be returned by a query started by the user. The keyword DEFAULT indicates that no value is defined by this profile and is equivalent to NO LIMIT ROWS.

If a numeric value or the keyword UNLIMITED is specified then this value will be used even when the SET QUERY LIMIT ROWS statement is present in the session, or when the logical name RDMS$BIND_QG_REC_LIMIT is defined.

NO LIMIT ROWS is the default.

LIMIT TIME
NO LIMIT TIME
LIMIT TIME sets the maximum elapsed time that can be used by the query compiler. The keyword DEFAULT indicates that no value is defined by this profile and is equivalent to NO LIMIT TIME.

If a numeric value or the keyword UNLIMITED is specified then this value will be used even when the SET QUERY LIMIT TIME statement is present in the session, or when the logical name RDMS$BIND_QG_TIMEOUT is defined.

NO LIMIT TIME is the default. Units can be specified as seconds or minutes.

NO DEFAULT TRANSACTION
NO LIMIT CPU TIME
NO LIMIT ROWS
NO LIMIT TIME
These options explicitly record the negated attribute setting. These clauses will remove the current setting of any clause being negated.

RENAME TO
Changes the name of the profile being altered. See the RENAME Statement for further discussion.

SET FLAGS '<string>'
NO SET FLAGS
Specifies whether or not a database system flag is set. The value is a quoted string of keywords separated by commas. See the SET FLAGS Statement for a list of keywords that can be set.

TRANSACTION MODES
TRANSACTION MODES provides the list of allowable transactions for this user. Please see the SET TRANSACTION MODES clause of the CREATE DATABASE and ALTER DATABASE statements for more details of txn-modes.

The transaction modes specified may include modes disabled for all database users by CREATE, IMPORT, or ALTER DATABASE statements. However, only the subset allowed by both profile and database settings will be used. For instance, if the database specifies (READ ONLY, SHARED READ, PROTECTED READ) and the profile specifies (READ ONLY, SHARED), the session will be allowed the subset (READ ONLY, SHARED READ).

See the description in the CREATE PROFILE Statement for all other attributes supported by ALTER PROFILE.

Usage Notes

- You must have SECURITY privilege on the database or OpenVMS SECURITY privilege to alter a profile.
- It is possible to restrict the transaction modes to READ ONLY using the default profile. Use caution in this case because it is possible that no user will have READ WRITE access to undo such a definition. In this case, you can define the logical name RDMS$SET_FLAGS to the value PROFILE_OVERRIDE to allow a suitably privileged user to start a transaction without using the transaction mode restrictions in the default profile. Such a user must have database SECURITY privilege, possibly inherited from the OpenVMS SECURITY process privilege.
- The string associated with the SET FLAGS clause is saved with the created profile. Any user that has this assigned profile will implicitly execute SET FLAGS during session start.

  Please notice that some SET FLAGS keywords affect actions during database attach and so have no action when defined within a profile. For example, DATABASE_PARAMETERS, generates minimal effects in such cases.
- See the CREATE PROFILE Statement for further details.

Examples

Example 1

The following example changes a default transaction for an existing profile.

```
SQL> ALTER PROFILE DECISION_SUPPORT
cont>   DEFAULT TRANSACTION READ ONLY;
```

Example 2

The following example shows the creation of a profile with flags and the assigning of the profile to a user. When this user (SMITH) attaches to a database (ATTACH, CONNECT, DECLARE ALIAS), or uses SET SESSION AUTHORIZATION, a SET FLAGS statement will implicitly be executed using this string of keywords.

```
SQL> create profile SF_USER
cont> set flags
cont> 'old_cost_model,noindex_column_group,optimization_level(total_time)'
cont> ;
SQL>
SQL> alter user SMITH
cont> profile SF_USER;
SQL>
```

Example 3

Note that the CREATE PROFILE and ALTER PROFILE statements will validate any listed keywords in the SET FLAGs clause.

```
SQL> alter profile SF_USER
cont> set flags 'THIS_OLD_HOUSE'
cont> ;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-EXT_ERR, Oracle Rdb extension error
-RDMS-E-UNKAMBFLAG, 'THIS_OLD_HOUSE' is an unknown or ambiguous flag name
SQL>
```

Example 4

The clause NO SET FLAGS can be used to remove any flags associated with a profile. All users assigned that profile will no longer perform a SET FLAGS action during session start.

```
SQL> show profile SF_USER
     SF_USER
     Flags: "old_cost_model,noindex_column_group,optimization_level(total_time)"
SQL> alter profile SF_USER
cont> no set flags;
SQL> show  profile SF_USER
     SF_USER
SQL>
```

ALTER ROLE Statement

Allows you to change the role name or add a comment to a role.

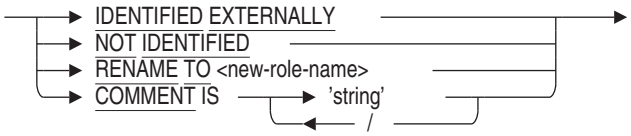
Environment

You can use the ALTER ROLE 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-role-opts =



Arguments

COMMENT IS 'string'
Adds a comment about the role. SQL displays the text of the comment when it executes a SHOW ROLES statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

IDENTIFIED EXTERNALLY
NOT IDENTIFIED
Specifies whether SQL should inherit roles from the operating system. If you specify one of these clauses, you must specify the same clause as was specified when the role was created. You cannot use the ALTER ROLE statement to change roles from IDENTIFIED EXTERNALLY to NOT IDENTIFIED or from NOT IDENTIFIED to IDENTIFIED EXTERNALLY.

The IDENTIFIED EXERNALLY clause indicates that SQL inherits the roles defined by the facilities of the operating system, such as OpenVMS rights identifiers.

The NOT IDENTIFIED clause indicates that SQL does not inherit any roles defined by the facilities of the operating system; instead, the role is private to the database.

RENAME TO new-role-name
Changes an existing role name to a new role name without changing the privileges granted to the role. You might change the name of a role that corresponds to a department name when the department is renamed. For example, if the personnel department is renamed human resources, you might change the role used by that department from PERSONNEL to HUMAN_ RESOURCES. The new role name must not already exist in the database. The old role name is removed from the database when the transaction is committed. The old role name can be re-created and reused, if desired. If the new role name is identified externally, then it must exist as an operating system group or rights identifier.

See the RENAME Statement for further discussion.

role-name
The name of an existing role (such as one created with the CREATE ROLE statement).

Usage Notes

- You must have the SECURITY privilege on the database or the OpenVMS SECURITY privilege to alter a role.
- The SHOW PROTECTION and SHOW PRIVILEGE statements will display the new role name created by the ALTER ROLE statement.
- If you issue the RENAME clause for a role identified externally, then the new role name must exist at the system level.

Example

Example 1: Renaming a Role

```
SQL> -- Change the name of the role from WRITER to DOCUMENTATION.
SQL> -- Any privileges granted to the role WRITER are transferred to the role
SQL> -- DOCUMENTATION. The role WRITER is deleted from the database.
SQL> ALTER ROLE WRITER
cont> RENAME TO DOCUMENTATION;
SQL> SHOW ROLES;
Roles in database with filename mf_personnel.rdb
     DOCUMENTATION
```

ALTER SEQUENCE Statement

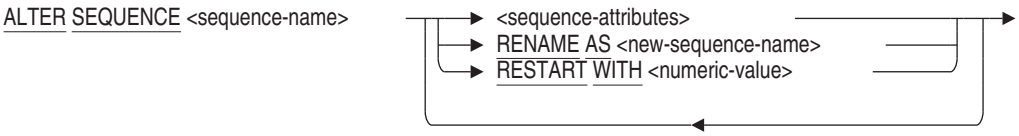
Alters a sequence. A sequence is a database object from which multiple users can generate unique integers. You can use sequences to automatically generate primary key values.

Environment

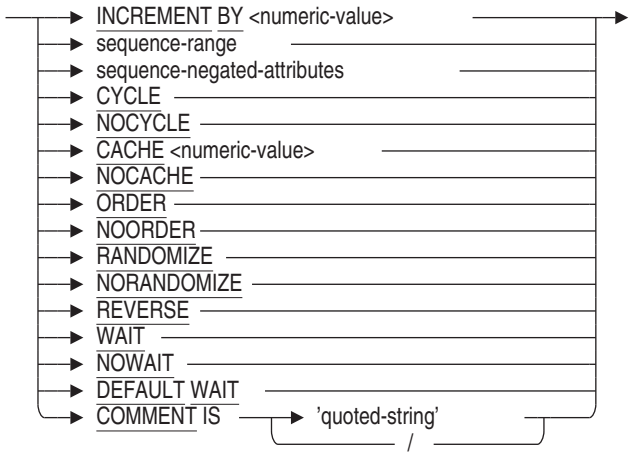
You can use the ALTER SEQUENCE statement:

- 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

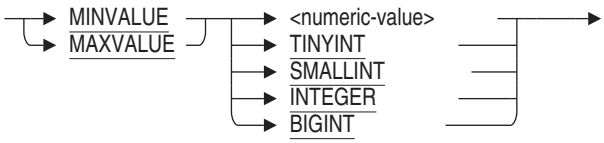
Format



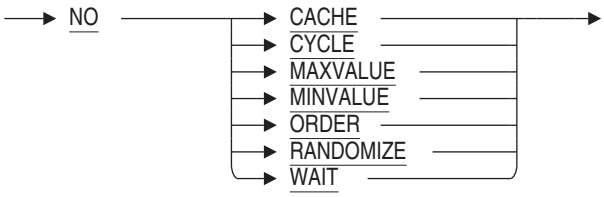
sequence-attributes =



sequence-range =



sequence-negated-attributes =



Arguments

CACHE numeric-value
NOCACHE

The CACHE clause specifies how many values of the sequence Oracle Rdb should preallocate and keep in memory for faster access. The mumeric value must be a value between 2 and 2147483647. You cannot cache more values than will fit in a given cycle of sequence numbers; thus, the maximum value allowed for the CACHE clause must be less than the value resulting from the following formula:

```
(MAXVALUE-MINVALUE)/ABS(INCREMENT)
```

You can alter the CACHE value if it is currently a value of 2 or higher. When you alter the CACHE value, existing users of the sequence continue to use the original setting. You can use the SET FLAGS 'SEQ_CACHE' statement to adjust the cache size for a single process. See the See the SET FLAGS Statement for details.

If NOCACHE is currently enabled or the CACHE value is 1, you can alter the CACHE value, but may have to wait until other users of the sequence have released locks on it. (Note that CACHE 1 is equivalent to NOCACHE.) See the Usage Notes for details.

A cache for a given sequence is populated at the first request for a number from that sequence, and whenever a value is requested when the cache is empty. If a system failure occurs, or when the cache is released any unfetched values will be discarded. The maximum number of lost values is equal to the current cache size. This may be the value specified by CACHE or by the SET FLAGS SEQ_CACHE option.

The NOCACHE clause specifies that values will be allocated one at a time. This will require more I/O to the Rdb root file than using a CACHE value.

Note that even after you alter the CACHE value, users who were using the sequence at the time you altered the CACHE will continue to use the original setting.

COMMENT IS 'string'
Adds a comment about the sequence. SQL displays the text of the comment when it executes a SHOW SEQUENCE statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

CYCLE
NOCYCLE
The CYCLE clause specifies that the sequence is to continue generating values after reaching either the MINVALUE or MAXVALUE. After an ascending sequence reaches the MAXVALUE, the sequence starts again from its MINVALUE. After a descending sequence reaches its MINVALUE, the sequence starts again at its MAXVALUE. The NOCYCLE clause (NO CYCLE is also accepted) specifies that the sequence should not continue generating values after reaching either its minimum or maximum value. An error is generated if an attempt is made to increment the sequence beyond its limits.

Note that even after you alter the CYCLE clause, those who were using the sequence at the time you altered the CYCLE will continue to use the original setting.

INCREMENT BY numeric-value
Specifies the size of the increment and the direction (ascending or descending) of the sequence. This numeric value must be in the range -2147483648 through 2147483647, excluding 0. The absolute value must be less than the difference between MAXVALUE and MINVALUE. A negative value specifies a descending sequence; a positive value specifies an ascending sequence. If the existing value is positive, then the new value must also be positive. Likewise, if the existing value is negative, then the new value must also be negative. That is, you cannot change a sequence from ascending to descending or from descending to ascending.

MAXVALUE numeric-value
NOMAXVALUE
The MAXVALUE clause specifies the maximum BIGINT value that the sequence can generate. For an ascending sequence, the new maximum value must be greater than or equal to the existing RDB$NEXT_SEQUENCE_VALUE. For a descending sequence, the new maximum value must be greater than or equal to the existing MAXVALUE. This ensures that the MAXVALUE is not less than any currently issued values. In addition, the numeric value must be between -9223372036854775808 and 9223372036854775807. The MAXVALUE must be greater than the value specified with the MINVALUE clause. The NOMAXVALUE clause (NO MAXVALUE also accepted) specifies that the maximum value for an ascending sequence is 9223372036854775807, and -1 for a descending sequence. The NOMAXVALUE (or NO MAXVALUE) clause is the default.

MAXVALUE TINYINT
MAXVALUE SMALLINT
MAXVALUE INTEGER
MAXVALUE BIGINT
SQL allows the keyword TINYINT, SMALLINT, INTEGER and BIGINT to follow MAXVALUE instead of a numeric value. This allows easy range setting for sequences used with these data types. The value supplied will be the largest positive value that can be assigned to this data type.

MINVALUE numeric-value
NOMINVALUE
The MINVALUE clause specifies the minimum BIGINT value that the sequence can generate. For an ascending sequence, the new minimum value must be less than or equal to the existing MINVALUE. For a descending sequence, the new minimum value must be less than or equal to the existing RDB$NEXT_SEQUENCE_VALUE. This prevents the minimum value from being greater than any currently issued values. In addition, the numeric value must be equal to or greater than -9223372036854775807. The MINVALUE must be less than the value specified with the MAXVALUE clause. The NOMINVALUE clause (NO MINVALUE also accepted) specifies that the minimum value for an ascending sequence is 1, and -9223372036854775807 for a descending sequence.

The NOMINVALUE (or NO MINVALUE) clause is the default.

MINVALUE TINYINT
MINVALUE SMALLINT
MINVALUE INTEGER

MINVALUE BIGINT
SQL allows the keyword TINYINT, SMALLINT, INTEGER and BIGINT to follow MINVALUE instead of a numeric value. This allows easy range setting for sequences used with these data types. The value supplied will be the smallest negative value that can be assigned to this data type.

ORDER
NOORDER
The ORDER clause specifies that sequence numbers are guaranteed to be assigned in order for each requesting process, thus maintaining a strict history of requests. The NOORDER clause (NO ORDER also accepted) specifies that sequence numbers are not guaranteed to be generated in order of request.

RANDOMIZE
NORANDOMIZE
The RANDOMIZE clause specifies that the sequence numbers are to be returned with a random value in the most significant bytes of the BIGINT value. This allows unique values to be generated that have a random distribution. When you specify the NORANDOMIZE clause (NO RANDOMIZE also accepted), sequence numbers are close in value to others created at the same time.

The advantage of the RANDOMIZE clause is that updates to columns of a stored index to which these values are written occur in different locations in the index structure and so may improve concurrent access for large indexes as leaf nodes in different parts of the index can be updated independently. In contrast, the sequence numbers generated when you specify the NORANDOMIZE clause are likely to be close in numeric value to other sequence values generated at the same time. This may cause index updates to occur in the same or nearby index nodes, which may lead to contention in one part of the sorted index.

The full range of values in the BIGINT value returned for the sequence are used; therefore, the NOMAXVALUE and NOMINVALUE clauses must be specified (or defaulted to) for the sequence definition. The most significant bits of the BIGINT value are set to a randomly generated positive value. A generated distinct value is returned in the least significant 32 bits so that uniqueness is guaranteed. If you also specify the CYCLE clause, then only the least significant 32 bits are cycled. When a query is performed on the column RDB$NEXT_SEQUENCE_VALUE in the RDB$SEQUENCES table, only the generated value of the least significant bits is returned, because the most significant bits are not assigned until the NEXTVAL pseudo column is referenced.

If you specify RANDOMIZE, you cannot also specify ORDER, MAXVALUE, or MINVALUE.

RENAME AS
Changes the name of the sequence being altered. See the RENAME Statement for further discussion. If the new name is the name of a synonym then an error will be raised.

The new name must not exist as the name of an existing sequence, synonym, table or view. You may not rename a system sequence.

The RENAME AS clause requires synonyms be enabled for this database. Refer to the ALTER DATABASE Statement SYNONYMS ARE ENABLED clause. Note that these synonyms may be deleted if they are no longer used by database definitions or applications.

RESTART
The RESTART clause allows the database administrator to reset the sequence to the initial starting value as supplied by the START WITH clause of the CREATE SEQUENCE statement or the RESTART WITH clause of the ALTER SEQUENCE statement.

Once the ALTER SEQUENCE statement is successfully committed, applications that use the sequence will start with a value based on the restarted value.

RESTART WITH
The RESTART WITH clause allows the database administrator to reset the sequence to a specified value. The value must be within the range of MINVALUE and MAXVALUE. This command requires exclusive access to the sequence. Once the ALTER SEQUENCE statement is successfully committed, applications that use the sequence will start with a value based on the restarted value.

Note

The TRUNCATE TABLE statement issued for a table with an IDENTITY column implicitly executes an ALTER SEQUENCE...RESTART WITH process on the sequence, applying the MINVALUE if it is an ascending sequence or MAXVALUE if it is a descending sequence.

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

WAIT
NOWAIT
DEFAULT WAIT
Specifies which wait state is used when a reference to NEXTVAL is used. A reference to NEXTVAL for a sequence may require synchronization with other users of the sequence. When you specify DEFAULT WAIT the wait state (WAIT or NOWAIT) of the current transaction is used. This may mean that no waiting is performed during a NOWAIT transaction (NO WAIT is also accepted).

If you specify WAIT (the default) for the sequence, then regardless of the wait state set for the current transaction, all synchronization waits for the next value. This is the recommended setting if the application uses NOWAIT transactions. The current WAIT timeout interval defined for the transaction or database is used.

If you specify NOWAIT for the sequence, then regardless of the current transaction setting, all synchronization will not wait for the next value.

Note that even after you alter the WAIT value, users who were using the sequence at the time you altered WAIT will continue to use the original setting.

Usage Notes

- You must have the ALTER privilege on the sequence to alter a sequence.
- If another user holds an exclusive lock on a sequence when you attempt to alter the sequence, your process will wait to execute the statement until the other user commits or rolls back his or her transaction. An exclusive lock is placed on a sequence when a user is altering any of the following attributes:
  - INCREMENT BY
  - MINVALUE to NOMINVALUE
  - NOMINVALUE to MINVALUE
  - MAXVALUE to NOMAXVALUE
  - NOMAXVALUE to MAXVALUE
  - NOCACHE to CACHE
  - ORDER to NOORDER
  - NOORDER to ORDER

- The value for the RDB$LAST_ALTERED column in the RDB$SEQUENCES system relation is updated by the ALTER SEQUENCE command. The value of RDB$NEXT_SEQUENCE_VALUE is not altered by this command.
- Note that sequences with the CYCLE attribute by definition will repeat values that were delivered perviously. Therefore, such sequences are not guaranteed to return unique values.
- The following are equivalent clauses: NOMAXVALUE and NO MAXVALUE, NOMINVALUE and NO MINVALUE, NOCYCLE and NO CYCLE, NOORDER and NO ORDER, NORANDOM and NO RANDOM, NOWAIT and NO WAIT. This was implemented to stay in sync with the SQL Standard.

Examples

Example 1: Altering a sequence

```
SQL> -- Show current sequence definition:
SQL> --
SQL> SHOW SEQUENCE EMPIDS
     EMPIDS
 Sequence Id: 1
 Initial Value: 1
 Minimum Value: 1
 Maximum Value: 9223372036854775787
 Next Sequence Value: 1
 Increment by: 1
 Cache Size: 20
 No Order
 No Cycle
```

```
 No Randomize
SQL> --
SQL> -- Alter the sequence.
SQL> --
SQL> ALTER SEQUENCE EMPIDS
cont> MINVALUE 0
cont> MAXVALUE 2000
cont> CACHE 30
cont> ORDER
cont> CYCLE;
SQL> --
SQL> -- Show new definition.
SQL> --
SQL> SHOW SEQUENCE EMPIDS
     EMPIDS
 Sequence Id: 1
 Initial Value: 1
 Minimum Value: (none)
 Maximum Value: 2000
 Next Sequence Value: 1
 Increment by: 1
 Cache Size: 30
 Order
 Cycle
 No Randomize
```

Example 2: Reset the sequence to a specified value

```
SQL> show sequence NEW_EMPLOYEE_ID
     NEW_EMPLOYEE_ID
 Sequence Id: 1
 Initial Value: 472
   .
   .
   .
SQL>
SQL> alter sequence NEW_EMPLOYEE_ID
cont> restart with 500;
SQL>
SQL> show sequence NEW_EMPLOYEE_ID
     NEW_EMPLOYEE_ID
 Sequence Id: 1
 Initial Value: 500
   .
   .
   .
SQL>
```

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
- Whether logging the transaction containing the ALTER statement is journaled to the RUJ and AIJ files.

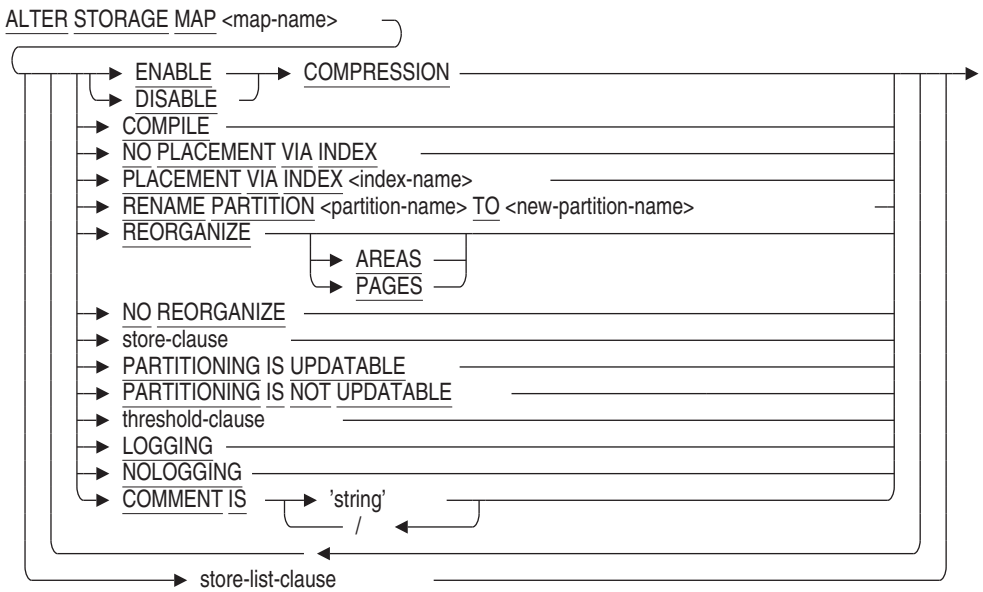
Environment

You can use the ALTER STORAGE MAP statement:

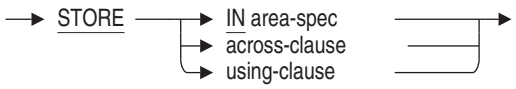
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

Format

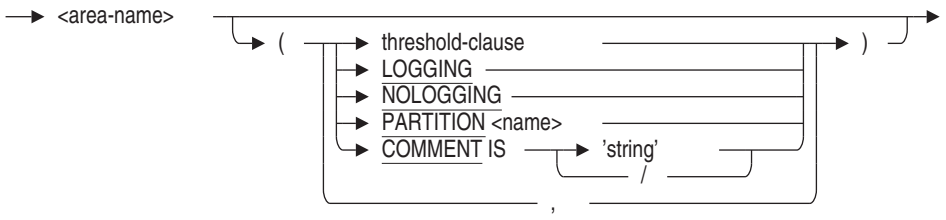
ALTER STORAGE MAP Statement



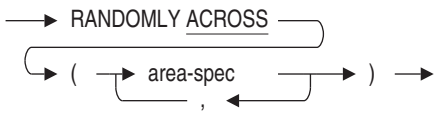
store-clause =



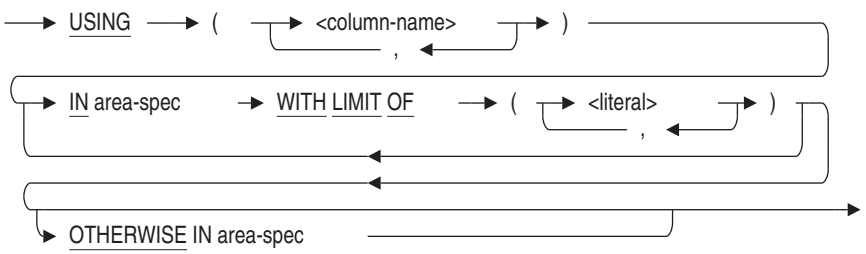
area-spec =



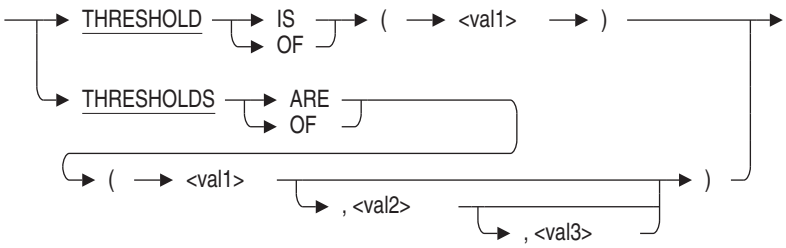
across-clause =



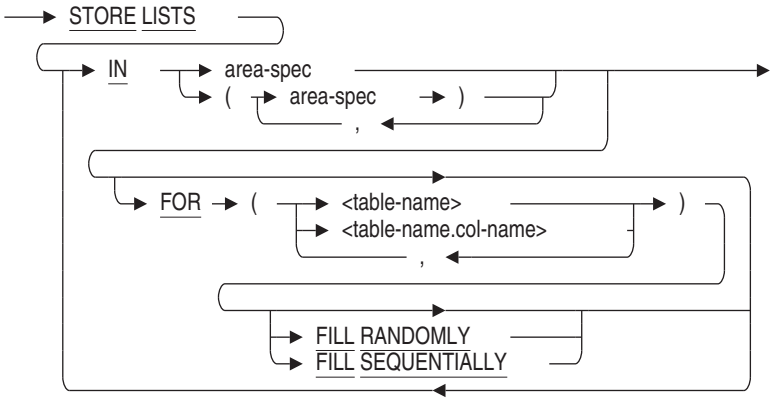
using-clause =



threshold-clause =



store-lists-clause =



Arguments

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.

COMMENT IS 'string'
Adds or alters a comment about the storage map. SQL displays the text of the comment when it executes a SHOW STORAGE MAPS statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

COMPILE
Creates a SQL mapping routine that matches the WITH LIMIT OF clause for the storage map. The routine is automatically created in the system module RDB$STORAGE_MAPS (use SHOW SYSTEM MODULES to view). The storage map name is used to name the mapping routine (use SHOW SYSTEM FUNCTIONS to view).

Note

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

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

The mapping routine returns the following values:

- Zero (0) if the storage map is defined as RANDOMLY ACROSS. This routine is just a descriptive place holder.
- Positive value representing the storage map number (the same value as stored in RDB$ORDINAL_POSITION column of the RDB$STORAGE_MAP_AREAS table). These values can be used with the PARTITION clause of the SET TRANSACTION...RESERVING clause to reserve a specific partition prior to inserting the row.
- A value of -1 if the storage map has no OTHERWISE clause. This indicates that the row cannot be inserted because it does not match any of the WITH LIMIT OF clauses.

ENABLE COMPRESSION
DISABLE COMPRESSION
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.

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.

If the set of areas is filled randomly, lists are stored across multiple areas. This is the default. Random filling benefits 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.

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 a , do not use this statement unless all areas specified 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.

LOGGING
The LOGGING clause specifies that the ALTER STORAGE MAP statement should be logged in the recovery-unit journal file (.ruj) and after-image journal file (.aij). The LOGGING clause is the default.

NOLOGGING
The NOLOGGING clause specifies that the ALTER STORAGE MAP statement should not be logged in the recovery-unit journal file (.ruj) and after-image journal file (.aij).

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. 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.

NO REORGANIZE
Disables the reorganize action for PARTITIONING IS NOT UPDATABLE.

PAGES
Specifies that the target of the data reorganization is database pages. All rows are checked to determine whether 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.

PARTITION name
Names the partition. The name can be a delimited identifier if the dialect or quoting rules are set to SQL92, SQL99 or SQL2011. Partition names must be unique within the storage map. If you do not specify this clause, Oracle Rdb generates a default name for the partition.

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 Rdb Guide to Database Design and Definition* for more information regarding partitioning.

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 as well as various optimizations for subsequent ALTER STORAGE MAP statements.

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

RENAME PARTITION partition-name TO new-partition-name
Specifies a new name for an existing storage map partition.

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 Rdb Guide to Database Design and Definition*.

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

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.

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 the default storage area for lists. This should be the LIST STORAGE AREA specified on CREATE DATABASE, or if none, the DEFAULT STORAGE AREA, or if none, then it will be RDB$SYSTEM.

For more information, see the CREATE STORAGE MAP Statement.

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. You cannot alter the thresholds for any storage areas which are part of the storage map. Only specify this clause for storage areas being added to the storage area by the ALTER STORAGE MAP statement.

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 Rdb Guide to Database Design and Definition*. See the CREATE STORAGE MAP Statement for a description of the THRESHOLDS clause.

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 complete 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. 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 following notes describe the behavior of the REORGANIZE clause:
  - If storage areas were named in the original storage map but are not now named in the ALTER STORAGE MAP STORE clause, then those rows will be moved from those areas and stored according to the new map definition. The moved row from this table will be deleted from the no longer referenced storage areas.
  - If the new storage map definition specifies the REORGANIZE AREAS clause, Oracle Rdb 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, Oracle Rdb 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 for the limits.
  - 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 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 a storage area, you cannot delete that area. 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.
- ALTER STORAGE MAP supports adding areas to a strictly partitioned map without requiring a rebuild of the table if they are appended to the end of the old map. The following example shows that no database I/O is performed to restructure the table.

```
SQL> set flags 'stomap_stats';
SQL> create table T1 (C1 integer);
SQL> create storage map S1
cont>     for T1
cont>     partitioning is not updatable
cont>     store using (C1)
cont>         in JOBS with limit of (100)
cont>         in DEPARTMENTS with limit of (200)
cont>         in EMP_INFO with limit of (300);
~As: create storage map "S1"
~As: Table "T1" (sys=0, rest=0, tmptbl=0)
.
.
.
SQL> -- add new area to the end of the map
SQL> -- no rebuild required
SQL> alter storage map S1
cont>     partitioning is not updatable
cont>     store using (C1)
cont>         in JOBS with limit of (100)
cont>         in DEPARTMENTS with limit of (200)
cont>         in EMP_INFO with limit of (300)
cont>         in EMPIDS_LOW with limit of (400);
~As: starting map restructure...
~As: reads: async 0 synch 0, writes: async 0 synch 0
.
```

```
.
.
SQL> -- add new area to the end of the map
SQL> -- no rebuild required
SQL> alter storage map S1
cont>     partitioning is not updatable
cont>     store using (C1)
cont>         in JOBS with limit of (100)
cont>         in DEPARTMENTS with limit of (200)
cont>         in EMP_INFO with limit of (300)
cont>         in EMPIDS_LOW with limit of (400);
~As: starting map restructure...
~As: reads: async 0 synch 0, writes: async 0 synch 0
```

The restructuring can only be eliminated if the original map omitted the OTHERWISE clause and if the new map specifies all existing partitions in the same order with the same WITH LIMIT clause.

- 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. If possible, Oracle Rdb moves the existing data to the appropriate storage area. However, it is possible that some rows cannot be moved because the partitioning key value violates the WITH LIMIT OF clause for the new final partition.
- 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 Rdb Guide to Database Design and Definition*.
- An existing storage map can be converted to a strictly partitioned storage map using the ALTER STORAGE MAP . . . PARTITIONING IS NOT UPDATABLE clause.

  This statement implicitly performs a reorganize operation on the base table, moving rows within the map if necessary, but at least scanning the storage areas to make sure all the stored data conforms to the storage map definition. This allows the Oracle Rdb optimizer to use this type of table efficiently when a sequential scan uses a subset of the storage areas.

  In many cases, the database administrator knows that a large table is already strictly partitioned, but it is prohibitive to reorganize the table. The amount of I/O alone might last several hours. Therefore, the database administrator can bypass the automatic reorganize operation performed by

the ALTER STORAGE MAP . . . PARTITIONING IS NOT UPDATABLE clause by using a NO REORGANIZE clause.

Because Oracle Rdb has not validated the table partitioning, there is a risk that rows may be missed by sequential scans. The database administrator must take this risk into account when using this clause. Oracle Corporation suggests that an ALTER STORAGE MAP . . . REORGANIZE operation be carried out as soon as practical.

When the NO REORGANIZE clause is used, Oracle Rdb records this information in the Oracle Rdb system relations. The SHOW STORAGE MAP statement will display informational text. Use SET FLAGS STOMAP_STATS to see a trace of the reorganize actions.

- The NO REORGANIZE clause is ignored unless used with PARTITIONING IS NOT UPDATABLE. This is because either no automatic reorganization is required, or a full rebuild of the table is needed to implement the new map structure.
- The REORGANIZE and NO REORGANIZE clause may not appear in the same ALTER STORAGE MAP command, as shown in the following example:

```
SQL> ALTER STORAGE MAP EMPLOYEES_MAP
cont>     PARTITIONING IS NOT UPDATABLE
cont>     NO REORGANIZE
cont>     REORGANIZE AREAS
cont>     STORE
cont>         USING (EMPLOYEE_ID)
cont>             IN EMPIDS_LOW
cont>                 WITH LIMIT OF ('00200')
cont>             IN EMPIDS_MID
cont>                 WITH LIMIT OF ('00400')
cont>             OTHERWISE IN EMPIDS_OVER;
%SQL-F-MULTSPECATR, Multiple specified attribute. "REORGANIZE" was specified
more than once
```

- The SET FLAGS option, STOMAP_STATS, will output an indication that NO REORGANIZE was used.
- The SHOW STORAGE MAPS statement will output an indication that NO REORGANIZE was used, as shown in the following example:

```
SQL> SHOW STORAGE MAPS EMPLOYEES_MAP
     EMPLOYEES_MAP
 For Table:             EMPLOYEES
 Placement Via Index:   EMPLOYEES_HASH
 Partitioning is:       NOT UPDATABLE
 Strict partitioning was not validated for this table
  . . .
```

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 for employees with 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: Enabling compression 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>
```

Example 4: Disabling Logging to the RUJ and AIJ files

```
SQL> ATTACH'FILENAME MF_PERSONNEL.RDB';
SQL> ALTER STORAGE MAP EMPLOYEES_MAP
cont>   STORE
cont>       USING (EMPLOYEE_ID)
cont>           IN EMPIDS_LOW
cont>               WITH LIMIT OF ('00200')
cont>           IN JOBS
cont>               (NOLOGGING)
cont>               WITH LIMIT OF ('00400')
cont>           OTHERWISE IN EMPIDS_OVER;
%RDB-W-META_WARN, metadata successfully updated with the reported warning
-RDMS-W-DATACMIT, unjournaled changes made; database may not be recoverable
```

Example 5: Disabled Area Scan for PARTITIONING IS NOT UPDATABLE

When a storage map is altered to be NOT UPDATABLE a REORGANIZE scan is implicitly executed to check that all rows are in the correct storage area according to the WITH LIMIT OF clauses in the storage map. This scan can be time consuming, and an informed database administrator may know that the data already conforms fully to the storage map. The NO REORGANIZE clause is used in the following example to avoid the extra I/O. The database administrator must understand that use of this clause might lead to incorrect query results (for sequential scans) if the storage map does not reflect the correct row mapping.

```
SQL> SET FLAGS 'stomap_stats';
SQL> ALTER STORAGE MAP EMPLOYEES_MAP
cont>     PARTITIONING IS NOT UPDATABLE
cont>     NO REORGANIZE
cont>     STORE
cont>         USING (EMPLOYEE_ID)
cont>             IN EMPIDS_LOW
cont>                 WITH LIMIT OF ('00200')
cont>             IN EMPIDS_MID
cont>                 WITH LIMIT OF ('00400')
cont>             OTHERWISE IN EMPIDS_OVER;
~As: starting map restructure...
~As: REORGANIZE needed to preserve strict partitioning
~As: NO REORGANIZE was used to override scan
~As: reads: async 0 synch 21, writes: async 7 synch 3
SQL>
SQL> SHOW STORAGE MAPS EMPLOYEES_MAP
     EMPLOYEES_MAP
 For Table:             EMPLOYEES
 Placement Via Index:   EMPLOYEES_HASH
 Partitioning is:       NOT UPDATABLE
 Strict partitioning was not validated for this table
 Comment:        employees partitioned by "00200" "00400"
 Store clause:          STORE
             using (EMPLOYEE_ID)
                 in EMPIDS_LOW
                     with limit of ('00200')
                 in EMPIDS_MID
                     with limit of ('00400')
                 otherwise in EMPIDS_OVER
 Compression is:        ENABLED
SQL>
```

A subsequent ALTER STORAGE MAP . . . REORGANIZE statement will validate the partitioning, as shown in the following example:

```
SQL> ALTER STORAGE MAP EMPLOYEES_MAP
cont>     PARTITIONING IS NOT UPDATABLE
cont>     REORGANIZE;
~As: starting map restructure...
~As: starting REORGANIZE...
~As: reorganize AREAS...
~As: processing rows from area 69
~As: processing rows from area 70
~As: processing rows from area 71
~As: reads: async 408 synch 22, writes: async 3 synch 0
SQL>
```

Example 6: Redefining a SQL routine that matches the WITH LIMIT OF clause for the storage map

The ALTER STORAGE MAP command removes any old mapping routine and redefines it when either the STORE clause is used, or if the COMPILE option is used.

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

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

ALTER SYNONYM Statement

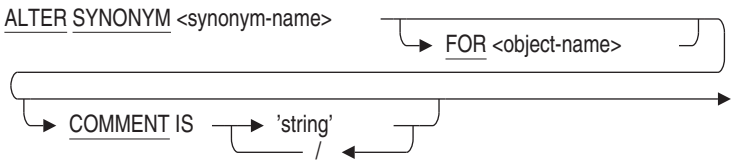
Alters a synonym definition.

Environment

You can use the ALTER SYNONYM statement:

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

Format



Arguments

COMMENT IS string
This clause can be used to add several lines of comment to the synonym object. The SHOW SYNONYM statement displays the comment.

This clause is equivalent to the COMMENT ON SYNONYM statement.

FOR object-name
You may change the synonym to reference a different database object; however, it must be of the same type. Oracle Rdb assumes that the object has the same or similar characteristics as the referenced object. The referenced object must exist in the database.

synonym-name
The name of an existing synonym you want to alter.

Usage Notes

- An error is generated if this statement is used on a database that has not been enabled for synonyms. See the ALTER DATABASE ... SYNONYMS ARE ENABLED clause.
- You must have database ALTER privilege in order to execute the ALTER SYNONYM statement.
- You must have REFERENCES privilege on the referenced object to alter a synonym for that object. Because domains do not have access control, no other privileges are required to alter synonyms for domains.
- You can alter synonyms for synonyms. Therefore, it is possible to create a cycle within a chain of synonyms. Oracle Rdb will detect this cycle and reject the definition.

Examples

Example 1: Adding a Comment

```
SQL> ALTER SYNONYM CASH
cont>   COMMENT IS 'use a different name to avoid confusion with'
cont>   /          'the domain MONEY';
```

Example 2: Using Multiple Synonyms and Changing the Referenced Table Using ALTER

The following example uses a synonym to reference a table. Later an empty version of the table can be created and the synonym altered to reference this new table. Although similar to using a view definition, the use of synonyms avoid the usage locking of a view. That is, to drop and create a new view requires that no other user references that view, however, the alter synonym does not require exclusive access to the table.

```
SQL> CREATE TABLE t_employees_0001 (...);
SQL> CREATE SYNONYM employees FOR t_employees_0001;
SQL> CREATE SYNONYM emps FOR employees;
SQL> CREATE TABLE t_employees_0002 LIKE t_employees_0001;
SQL> ALTER SYNONYM employees FOR t_employees_0002;
```

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. 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. See also the ALTER CONSTRAINT Statement and the DROP CONSTRAINT Statement.

When you execute this statement, SQL modifies the named column definitions in the table. All of the columns that you do not mention remain unchanged. 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 connect to the database after you commit the changes.

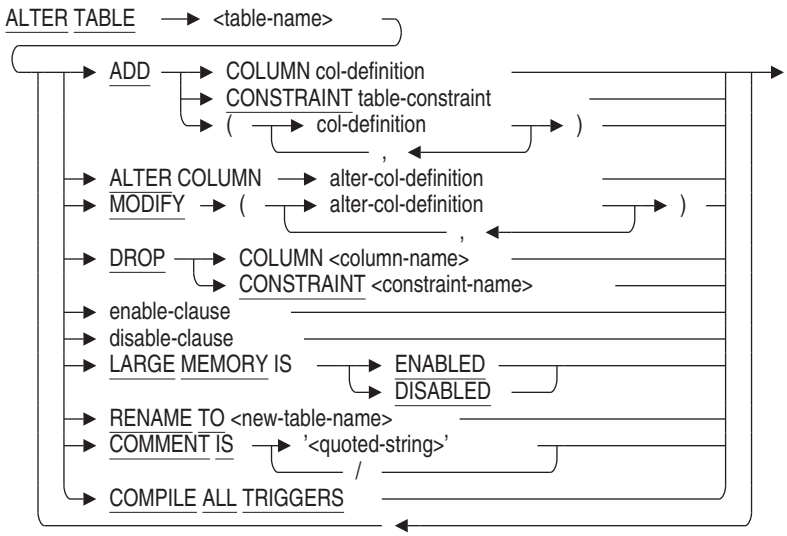
Environment

You can use the ALTER TABLE statement:

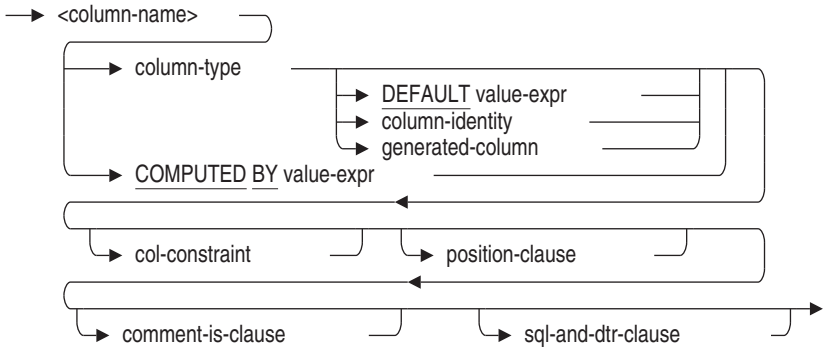
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

ALTER TABLE Statement

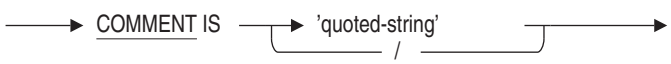
Format



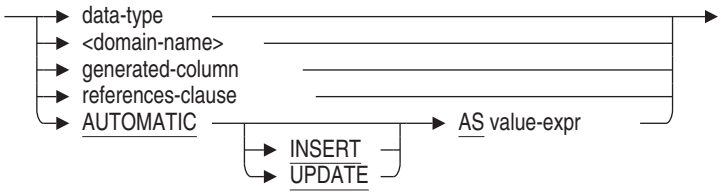
col-definition =



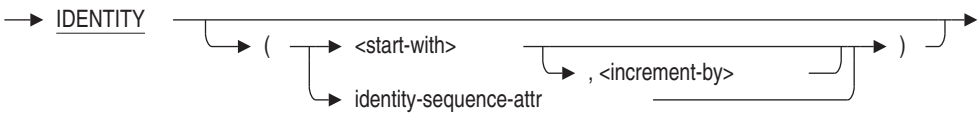
comment-is-clause =



column-type=

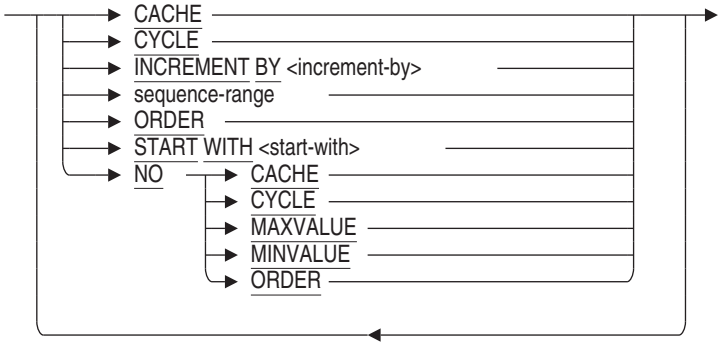


column-identity =



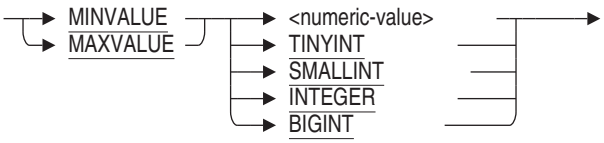
generated-column =

identity-sequence-attr =

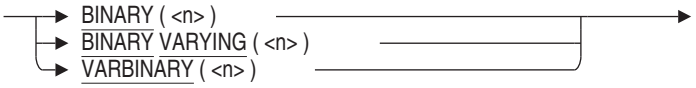


ALTER TABLE Statement

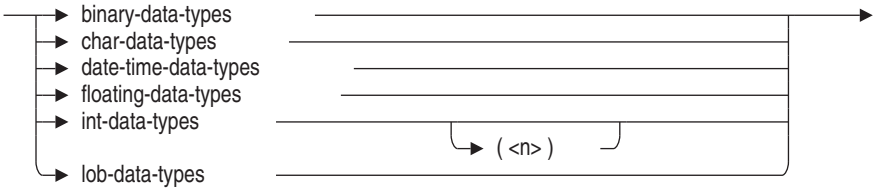
sequence-range =



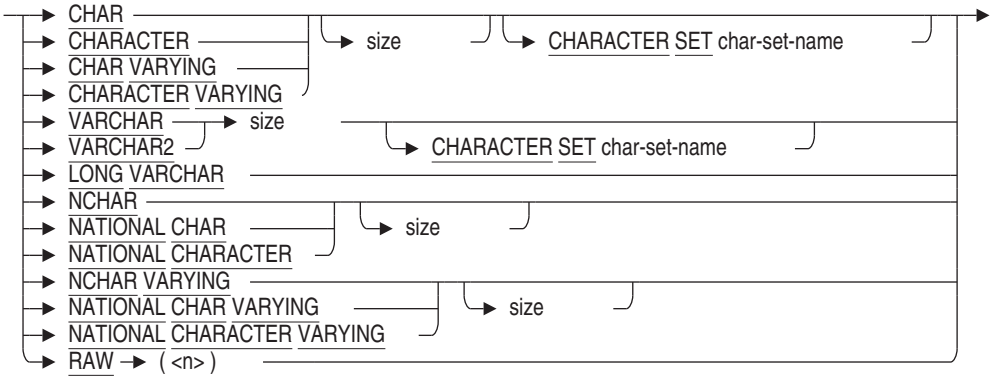
binary-data-types =



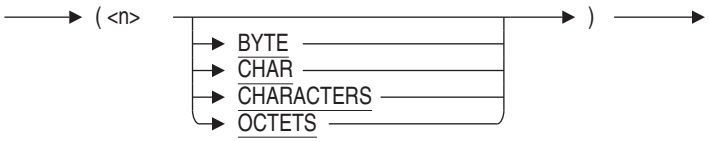
data-type =



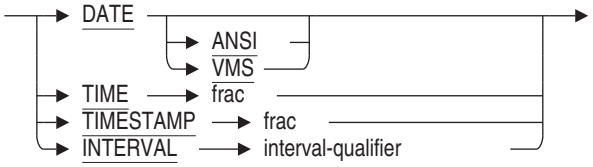
char-data-types =



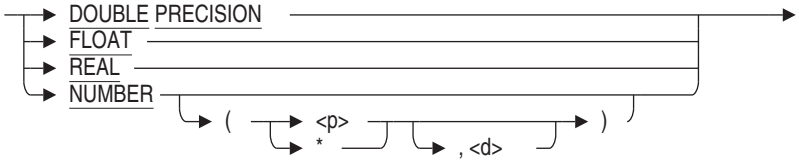
size =



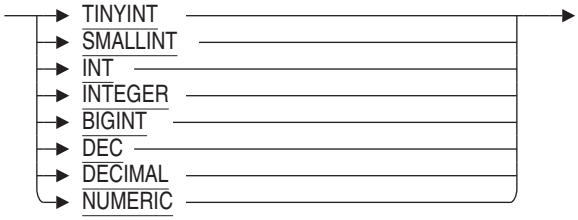
date-time-data-types =



floating-data-types =

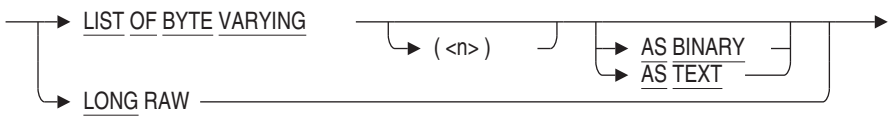


int-data-types =

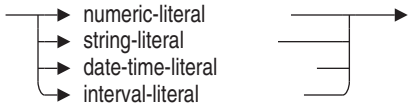


ALTER TABLE Statement

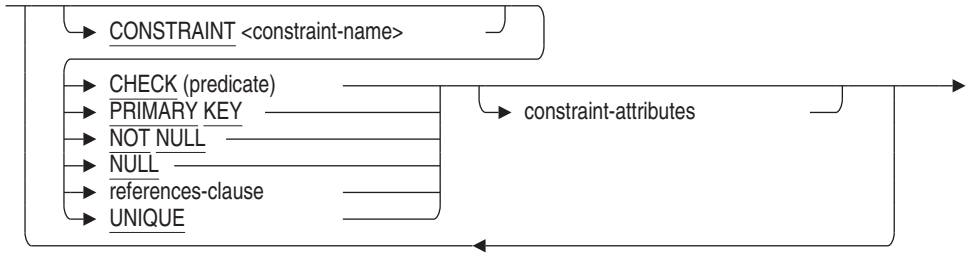
lob-data-types =



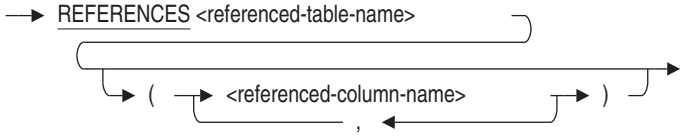
literal =



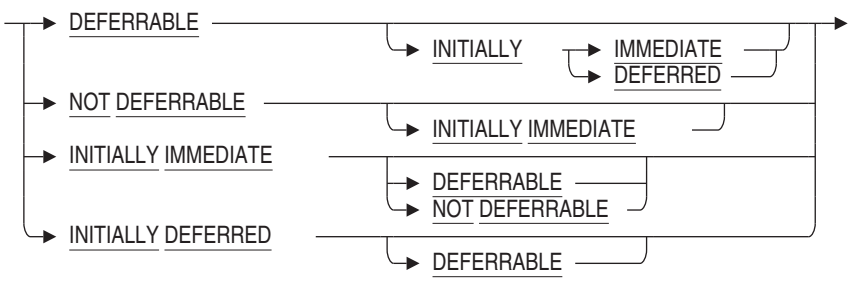
col-constraint=



references-clause =



constraint-attributes =



position-clause =

sql-and-dtr-clause =

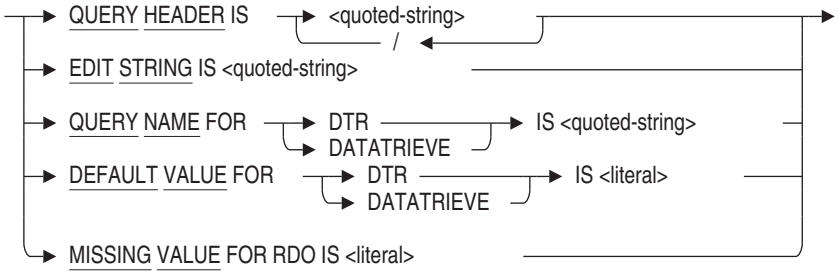
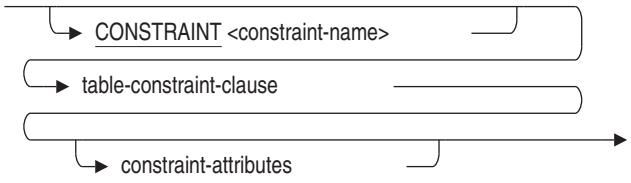
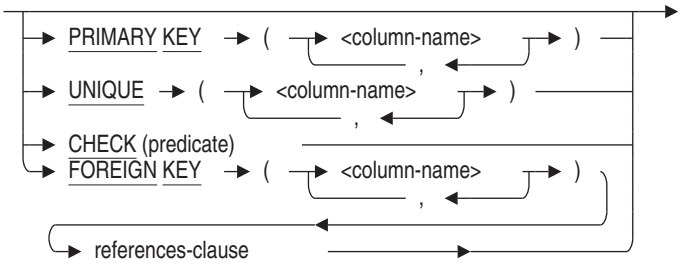


table-constraint =

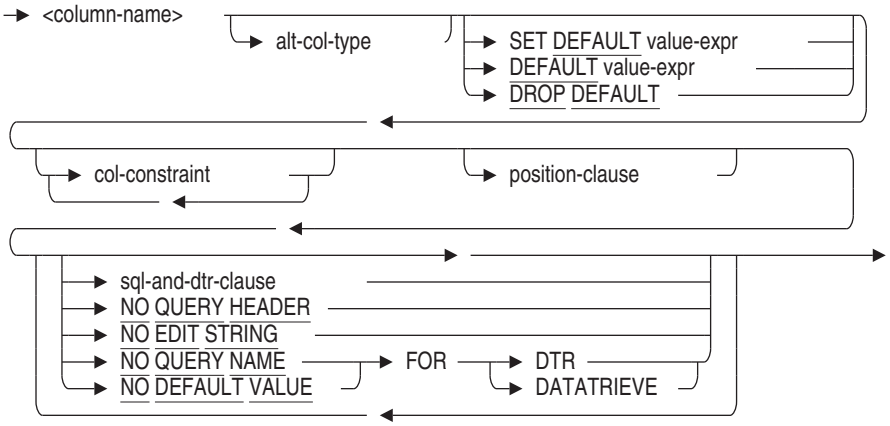


ALTER TABLE Statement

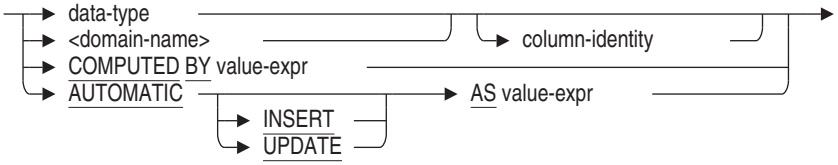
table-constraint-clause =



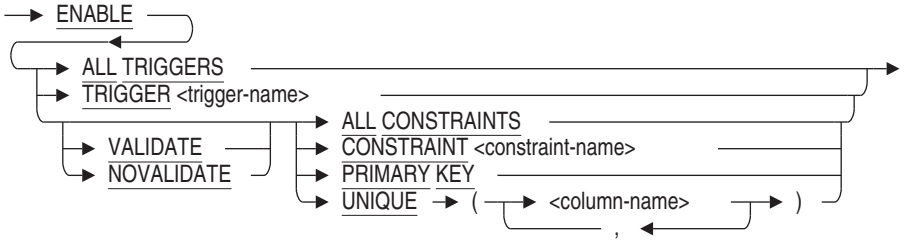
alter-col-definition =



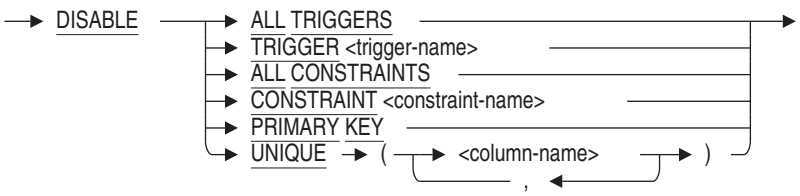
alt-col-type =



enable-clause =



disable-clause =



Arguments

ADD (...)
This alternate syntax is added for compatibility with Oracle Database.

ADD COLUMN col-definition
Creates an additional column in the table. SQL adds the column after all existing columns in the table unless the position-clause relocates the new column. The column definition specifies a data type or domain name, optional default value, optional column constraints, and optional formatting clauses.

The COLUMN keyword is optional.

ADD CONSTRAINT table-constraint
Adds a table constraint definition. The four types of table constraints are PRIMARY KEY, UNIQUE, CHECK, and FOREIGN KEY.

AFTER COLUMN column-name
BEFORE COLUMN column-name
Changes the normal field ordering of columns to make the displayed column ordering more readable. Note that this does not change the on-disk layout of the columns. By default, when neither of these clauses is specified, columns are positioned at the end of the table specified with the ALTER TABLE statement.

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.

AUTOMATIC AS value-expr
AUTOMATIC INSERT AS value-expr
AUTOMATIC UPDATE AS value-expr
These AUTOMATIC column clauses allow you to store special information when data is inserted into a row or a row is updated. For example, you can log application-specific information to audit activity or provide essential values, such as time stamps or unique identifiers for the data.

The assignment of values to these types of columns is managed by Oracle Rdb. The AUTOMATIC INSERT clause can be used to provide a complex default for the column when the row is inserted; it cannot be changed by an UPDATE statement. The AUTOMATIC UPDATE clause can be used to provide an updated value during an UPDATE statement. The AUTOMATIC clause is the default and specifies that the value expression should be applied during both INSERT and UPDATE statements. The column type is derived from the AS value-expr; using CAST allows a specific data type to be specified. However, this is not required and is rarely necessary.

You can define an AUTOMATIC INSERT column to automatically receive data during an insert operation. The data is stored like any other column, but the column is read-only. Because AUTOMATIC columns are treated as read-only columns, they cannot appear in the column list for an insert operation nor be modified by an update operation. AUTOMATIC UPDATE columns can have an associated default value that will be used when the row is inserted.

Suppose that you want to store the current time stamp of a transaction and supply a unique numeric value for an order number. In addition, when the row is updated (the order is altered), you want a new time stamp to be written to the LAST_UPDATED column. You could write an application to supply this information, but you could not guarantee the desired behavior. For instance, a user with access to the table might update the table with interactive SQL and forget to enter a new time stamp to the LAST_UPDATED column. If you use an AUTOMATIC column instead, it can be defined so that columns automatically receive data during an insert operation. The data is stored like any other column, but the column is read-only.

See the Usage Notes for more information on automatic columns.

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

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.

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;
```

column-name
The name of the column being added or modified.

COMPILE ALL TRIGGERS
This clause directs Oracle Rdb to re-compile all the triggers defined for the table to ensure that they are valid. If COMPILE ALL TRIGGERS is successful and any trigger was marked invalid and "Can be revalidated", then the invalid flag will be cleared.

Triggers can be marked "invalid" when a procedure, function or sequence is dropped using the CASCADE clause.

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.

constraint-attributes

Although the constraint attribute syntax, shown in Table 6–3, provides 11 permutations as required by the SQL Database Language standard, they equate to the followingn three options:

- INITIALLY IMMEDIATE NOT DEFERRABLE

  Specifies that evaluation of the constraint must take place when the INSERT, DELETE, or UPDATE statement executes. If you are using the SQL92, SQL99, SQL2011, MIA, ORACLE LEVEL1, ORACLE LEVEL2 or ORACLE LEVEL3 dialect, this is the default.

  This clause is the same as the NOT DEFERRABLE option provided in previous releases of Oracle Rdb.

- INITIALLY DEFERRED DEFERRABLE

  Specifies 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, this is the default constraint attribute. When using this default dialect, Oracle Rdb displays a deprecated feature message for all constraints defined without specification of one of the constraint attributes.

  This clause is the same as the DEFERRABLE option provided in previous releases of Oracle Rdb.

- INITIALLY IMMEDIATE DEFERRABLE

  Specifies that evaluation of the constraint be deferred (using the SET CONSTRAINT ALL statement or the SET TRANSACTION statement with the EVALUATING clause) but by default it is evaluated after the INSERT, DELETE, or UPDATE statement executes.

Table 6–3 Constraint Attributes Syntax Permutations and Equivalents

If You Specify This Clause:	It Defaults to This Clause:
Do not specify a clause	INITIALLY IMMEDIATE NOT DEFERRABLE
NOT DEFERRABLE	
INITIALLY IMMEDIATE	

(continued on next page)

Table 6–3 (Cont.) Constraint Attributes Syntax Permutations and Equivalents

If You Specify This Clause:	It Defaults to This Clause:
INITIALLY IMMEDIATE NOT DEFERRABLE NOT DEFERRABLE INITIALLY IMMEDIATE	
INITIALLY DEFERRED DEFERRABLE INITIALLY DEFERRED INITIALLY DEFERRED DEFERRABLE	INITIALLY DEFERRED DEFERRABLE
DEFERRABLE INITIALLY IMMEDIATE DEFERRABLE DEFERRABLE INITIALLY IMMEDIATE	INITIALLY IMMEDIATE DEFERRABLE

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 CONSTRAINT, DROP CONSTRAINT and 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. If you supply a constraint name with the CONSTRAINT clause, the name must be unique in the schema.

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 character 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 smaller capacity numeric data type then overflow errors may result at run time as Oracle Rdb attemps to convert the large value to the new data type.
- 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, or date/time (DATE ANSI, TIMESTAMP, TIME). If you attempt to change a DATE column to anything else, SQL returns an error message.

date-time-data-types
A valid SQL date-time data type. See Section 2.3.2 for more information on date-time data types.

DEFAULT value-expr
Provides a default value for a column if the row that is inserted does not include a value for that column.

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.

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.

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.

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 previously stored rows.

The following example shows that SQL stores the default value in the column when you add a column that specifies a default value.

```
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 a single 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*.

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, and 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 Oracle Database language.
- You are creating tables that do not need the advantages of domains.

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.

DROP DEFAULT
Deletes (drops) the default value of a column in a table.

enable-clause
disable-clause

Allows you to enable or disable all triggers, specified triggers, all constraints, specified constraints, a primary key, or a unique constraint, as described in the following list. By default, table and column constraints added during an alter table operation are enabled.

- DISABLE ALL TRIGGERS

  All triggers defined for the table are disabled. (No error is raised if no triggers are defined for this table.)

- ENABLE ALL TRIGGERS

  All triggers defined for the table are enabled. (No error is raised if no triggers are defined for this table.)

- DISABLE TRIGGER trigger-name

  The named trigger for this table is disabled. The named trigger must be defined on the table.

- ENABLE TRIGGER trigger-name

  The named trigger for this table is enabled. The named trigger must be defined on the table.

- DISABLE ALL CONSTRAINTS

  All table and column constraints for this table are disabled. (No error is raised if no constraints are defined on the table.)

- ENABLE ALL CONSTRAINTS

  All table and column constraints for this table are enabled. (No error is raised if no constraints are defined on the table.)

- DISABLE CONSTRAINT constraint-name

  The named constraint is disabled. The named constraint must be a table or column constraint for the table.

- ENABLE CONSTRAINT constraint-name

  The named constraint is enabled. The named constraint must be a table or column constraint for the table.

- DISABLE PRIMARY KEY

  The primary key for the table is disabled.

- ENABLE PRIMARY KEY

  The primary key for the table is enabled.

- DISABLE UNIQUE (column-name)

  The matching UNIQUE constraint is disabled. The columns listed must be columns in a unique constraint for the table.

- ENABLE UNIQUE (column-name)

  The matching UNIQUE constraint is enabled. The columns listed must be columns in a unique constraint for the table.

- VALIDATE and NOVALIDATE

  When a constraint is added or enabled with the ALTER TABLE statement, the default is to validate the table contents. The ENABLE NOVALIDATE option allows a knowledgeable database administrator to avoid the time and I/O resources required to revalidate the data when they know the data is valid.

  Note

  Oracle Corporation recommends that you use the RMU Verify command with the Constraint qualifier periodically to verify that your assumptions are correct if you use the ENABLE NOVALIDATE option.

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).

GENERATED ALWAYS AS IDENTITY

This clause is equivalent to the Oracle Rdb syntax IDENTITY [identity-attributes] and is added for compatibility with Oracle Database and the ANSI/ISO SQL Database Language Standard.

When defining a column, the data type of the column can be provided, as shown in the following example, and will result in an implicit CAST of the value-expression to that data type (or domain).

```
SQL> create domain SEQ_NO_DOM integer;
SQL>
SQL> create table SAMPLE
cont>     (seq_no SEQ_NO_DOM generated always as identity
cont>     !...
cont>     );
SQL>
SQL> show table (column) SAMPLE;
Information for table SAMPLE
```

```
Columns for table SAMPLE:
Column Name                    Data Type        Domain
-----------                    ---------        ------
SEQ_NO                         INTEGER
 Computed:      Generated always as Identity
    !...
SQL>
```

If the data type is omitted, then the default will be BIGINT.

GENERATED BY DEFAULT AS IDENTITY

This clause is similar to the GENERATED ALWAYS AS IDENTITY clause, with the exception that the application programmer may INSERT a value instead of having Oracle Rdb compute and store a value.

In contrast, the GENERATED ALWAYS clause is treated as a read-only column. Note: the database administrator can also use the SET FLAGS 'AUTO_OVERRIDE' statement to temporarily treat GENERATED ALWAYS columns as GENERATED BY DEFAULT columns.

If an explicit value is inserted by the application, then it is possible that the sequence associated with the IDENTITY column will generate a duplicate value. The application must be prepared to handle this case and Oracle Rdb cannot guarantee uniqueness of values in this column.

GENERATED ALWAYS AS (value-expr)

This clause is equivalent to the Oracle Rdb syntax AUTOMATIC INSERT AS value-expression and is added for compatibility with Oracle Database and the ANSI/ISO SQL Database Language Standard.

When defining a column, the data type of the column can be provided, as shown in the following example, and will result in an implicit CAST of the value-expression to that data type (or domain).

```
SQL> create table SAMPLE
cont>     (seq_no SEQ_NO_DOM generated always as identity
cont>     ,row_ts timestamp(2) generated always as ( current_timestamp )
cont>     !...
cont>     );
SQL>
SQL> show table (column) SAMPLE;
Information for table SAMPLE
```

```
Columns for table SAMPLE:
Column Name                    Data Type        Domain
-----------                    ---------        ------
SEQ_NO                         INTEGER
 Computed:      Generated always as Identity
ROW_TS                         TIMESTAMP(2)
 Computed:      Generated always as ( current_timestamp )
    !...

SQL>
```

If the column data type is not specified, then the data type will be derived from the value expression.

GENERATED BY DEFAULT AS (value-expr)

This clause is similar to the GENERATED ALWAYS clause, with the exception that the application programmer may INSERT a value instead of having Oracle Rdb compute and store a value.

In contrast, the GENERATED ALWAYS clause is treated as a read-only column. Note: the database administrator can also use the SET FLAGS 'AUTO_OVERRIDE' statement to temporarily treat GENERATED ALWAYS columns as GENERATED BY DEFAULT columns.

IDENTITY

Specifies that the column is to be a special read-only identity column. INSERT will evaluate this column and store a unique value for each row inserted. Only one column of a table may have the IDENTITY attribute. Oracle Rdb creates a sequence with the same name as the current table.

See the ALTER SEQUENCE Statement and the CREATE SEQUENCE Statement for more information.

* Increment-by

  An integer literal value that specifies the increment for the sequence created for the IDENTITY column. A negative value creates a descending sequence, and a positive value creates an ascending sequence. A value of zero is not permitted. If omitted the default is 1, that is an ascending sequence.

* Start-with

  An integer literal value that specifies the starting value for the sequence created for the IDENTITY column. If omitted the default is 1.

LARGE MEMORY IS ENABLED
LARGE MEMORY IS DISABLED
This clause allows LARGE MEMORY to be enabled or disabled. It will enable or disable the use of 64 bit memory for the temporary data structures and user data.

The LARGE MEMORY clause can only be used for a GLOBAL or LOCAL TEMPORARY table.

MAXVALUE TINYINT
MAXVALUE SMALLINT
MAXVALUE INTEGER
MAXVALUE BIGINT
SQL allows the keyword TINYINT, SMALLINT, INTEGER and BIGINT to follow MAXVALUE instead of a numeric value. This allows easy range setting for identity columns that are used with these data types. The value supplied will be the largest positive value that can be assigned to this data type.

MINVALUE TINYINT
MINVALUE SMALLINT
MINVALUE INTEGER
MINVALUE BIGINT
SQL allows the keyword TINYINT, SMALLINT, INTEGER and BIGINT to follow MINVALUE instead of a numeric value. This allows easy range setting for identity columns used with these data types. The value supplied will be the smallest negative value that can be assigned to this data type.

MODIFY (...)
This alternate syntax is added for compatibility with Oracle Database.

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

NULL
Specifies that NULL is permitted for the column. This is the default behavior. A column with a NULL constraint cannot also have a NOT NULL constraint within the same ALTER TABLE statement. However, no checks are performed for CHECK constraints, which may limit the column to non-null values.

The NULL constraint is not stored in the database and is provided only as a syntactic alternative to NOT NULL.

When used on ALTER TABLE . . . ALTER COLUMN this clause drops any NOT NULL constraints defined for the column.

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 primary key columns.

You cannot specify the primary key constraint for a computed column.

When used as a table constraint this clause must be followed by a list of column names. When used as a column constraint this clause applies to the named column of the 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.

references-clause

Specifies the name of the column or columns that are a unique key or primary key or in the referenced table. When the REFERENCES clause is used as a table constraint, the column names specified in the FOREIGN KEY clause become a foreign key for the referencing table.

When used as the column type clause, specifies that the type of the column be inherited from the PRIMARY KEY or UNIQUE index referenced. Both the data type and domain are inherited.

RENAME TO

Changes the name of the table being altered. See the RENAME Statement for further discussion. If the new name is the name of a synonym then an error will be raised.

The new name must not exist as the name of an existing table, synonym, sequence or view. You may not rename a system table.

The RENAME TO clause requires synonyms be enabled for this database. Refer to the ALTER DATABASE Statement SYNONYMS ARE ENABLED clause. Note that these synonyms may be deleted if they are no longer used by database definitions or applications.

SET DEFAULT default-value

Specifies a default value for the column.

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.

start-with
An integer literal value that specifies the starting value for the sequence created for the IDENTITY column. If omitted the default is 1.

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

UNIQUE
Specifies that values in the associated column must be unique.

Usage Notes

- You can revise a COMPUTED BY or AUTOMATIC column expression by the ALTER TABLE...ALTER COLUMN statement. A new row version is created for the table to accommodate changes in data type and length of the data. If the column is an AUTOMATIC AS definition, then the previously stored data will be converted to the new data type upon retrieval.

  Only the value expression can be altered; the type of computation cannot be changed. That is, a COMPUTED BY column cannot be changed to an AUTOMATIC AS column and an AUTOMATIC INSERT AS column cannot be changed to an AUTOMATIC UPDATE AS column.

- 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 complete 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.

- You can only use the ALTER TABLE statement to alter table definitions. Use the ALTER VIEW statement to modify 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 Rdb Guide to SQL Programming* for detailed information about stored procedure dependency types and how metadata changes can cause invalidation of stored procedures.
- If a column is part of an index, you cannot alter its data type unless the result type has the same type, length, precision, scale, character set, collating sequence, and so on. For example, the column EMPLOYEE_ID is defined in the table EMPLOYEE as ID_DOM. The following example shows that it can be altered when the same type is used.

```
SQL> show table (column) EMPLOYEES
Information for table EMPLOYEES

Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          ID_NUMBER
 Missing Value:
   .
   .
   .
SQL> alter table employees alter column employee_id 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-RELFLDNOC, field EMPLOYEE_ID in relation EMPLOYEES has not been changed

SQL> alter table employees alter column employee_id char(5);
SQL> show table (column) EMPLOYEES
Information for table EMPLOYEES
```

```
Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)
   .
   .
   .
SQL>
```

- You cannot delete a column in a table if:
  - That column is referred to by some other database object, such as a view, constraint or trigger.
  - An index is based on that column.
- You can alter the data type of a column with a referencing NOT NULL constraint without first deleting the constraint.
- If the data types are the same, the ALTER COLUMN clause can change an AUTOMATIC column to an updatable base column even if it has constraints and indices.
- You can alter a non-computed base column to be an AUTOMATIC column. The old data is retained and the column is made read-only.
- You can alter a non-computed base column to be a COMPUTED BY column. The old data will not be accessible (a warning is issued for interactive SQL), and references to that column will evaluate the COMPUTED BY expression. The ALTER TABLE statement will fail if indices reference this column.

  Altering the column back to a base or automatic column will allow older versions of the row data to be visible; any rows inserted while the column was a COMPUTED BY column will return a null value.
- You can use the IDENTITY syntax with the ALTER COLUMN clause.

  If the table has no existing IDENTITY column, a new sequence for the table will be created. You must ensure that use of IDENTITY will not generate existing values for the column because this would cause the insert operation to fail. Use the parameters in the IDENTITY syntax to specify an appropriate START WITH VALUE, or modify the sequence using the ALTER SEQUENCE command.

  If the table has an existing IDENTITY column, an error is displayed.
- If an IDENTITY column is converted to a base column, a COMPUTED BY column, or an AUTOMATIC column, the special sequence is automatically dropped.

- If a column has a DEFAULT (base column or AUTOMATIC UPDATE AS column) and it is converted to a COMPUTED BY, AUTOMATIC AS, or an AUTOMATIC INSERT AS column, the default value is removed because these types of columns are incompatible with DEFAULT.
- 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.
- 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 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 may return 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.

- Any trigger marked with the invalid flag will still be used by Oracle Rdb if at runtime it can be compiled successfully. However, only the COMPILE clause of ALTER TRIGGER statement, or the COMPILE ALL TRIGGERS clause of ALTER TABLE statement will clear the "invalid" flag.
- The following usage notes apply to constraints:
  - In addition to the ALTER privilege that is required to issue the ALTER TABLE statement, you must have the DROP privilege on the table to disable a constraint or trigger, and you must have the CREATE privilege on the table to enable a constraint or a trigger. You must hold the DBADM privilege on the database to specify the NOVALIDATE option to the enable-clause.
  - When a constraint is disabled, it is not evaluated by the INSERT, UPDATE, or DELETE statements.
  - When a previously disabled constraint is reenabled, the constraint is validated to ensure that all existing rows are valid.

- If you enable constraints during an alter table operation, and do not specify NOVALIDATE, each constraint will be validated. If the referential integrity constraint fails, then the alter table operation also fails and the failed constraint name will be reported.
- The RMU Verify command with the Constraint qualifier ignores any disabled constraint unless that constraint is named with the Constraint qualifier and the Constraint option. If the Constraint qualifier specifies all constraints (no options are specified), or if it specifies a specific table, all disabled constraints are ignored. This allows you to check a disabled constraint periodically without the need to reenable it, which might be useful if the overhead of checking the constraint during operating hours is too expensive, or if it is already being enforced by the application.

• The following usage notes apply to AUTOMATIC columns:

- When the column is omitted from an insert operation, a column default and an automatic column provide similar functions. However, there are distinctions, as follows:
  * AUTOMATIC columns cannot be referenced during an insert operation because they are read-only to applications.
  * AUTOMATIC columns can be active during an update operation.
  * When you use an AUTOMATIC column, you do not provide the data type for the column.
- Note the following differences between using COMPUTED BY columns and AUTOMATIC columns:
  * COMPUTED BY columns use no space in the row, AUTOMATIC columns do.
  * A COMPUTED BY column is evaluated when the row is fetched, such as when a SELECT, UPDATE, or DELETE statement references the column name. An AUTOMATIC column is evaluated during an INSERT or UPDATE statement. A calculated value is written to a column in the row and the value returned by a SELECT statement is the stored column value.

    For example, a column defined as COMPUTED BY CURRENT_ DATE returns the date when the query is executed. A selected column that is AUTOMATIC INSERT AS CURRENT_DATE returns the date when the INSERT was performed, which might be different from the date when the query is executed.

- Note the following differences between using an AUTOMATIC column and a trigger on the table:
  * In an insert operation, an AFTER INSERT trigger can provide this functionality. However, AUTOMATIC columns can help eliminate the overhead of a trigger and so simplify table management.
  * Trigger actions cannot modify a row being updated, because this leads to a recursive trigger action. AUTOMATIC UPDATE columns are evaluated prior to the trigger and constraint execution.
- You can create an index and constraints on an AUTOMATIC column. AUTOMATIC columns are identical to other columns, except that Oracle Rdb, not a user application, assigns the value.
- If the data written to the table with an AUTOMATIC column is incorrect, you can temporarily suspend the read-only attribute of the column by issuing the SET FLAGS 'AUTO_OVERRIDE' statement if you have the DBADMIN privilege on the database. Then you can execute an update query to correct the incorrect data. See the SET FLAGS Statement for more information and an example.
- The SET FLAGS option AUTO_OVERRIDE can be used to allow updates to selected AUTOMATIC columns during INSERT so that rows could be reloaded, or during UPDATE to adjust incorrectly stored values.
  * For the INSERT statement or REPLACE statement, 'AUTO_ OVERRIDE' allows assignment to any AUTOMATIC column, and any insert AUTOMATIC column omitted from the column list will be evaluated normally.
  * For the UPDATE statement 'AUTO_OVERRIDE' allows direct assignment of values to any AUTOMATIC column. No AUTOMATIC columns are evaluated.
- If the DEFAULT clause is used in an INSERT or UPDATE statement then one of the following will be applied:
  * When a new column is added that includes the DEFAULT attribute then all previously inserted rows are updated to include the DEFAULT as the value of the new column.

    If the new column is based on a domain which includes the DEFAULT attribute then, in the same fashion, the table is updated.

If the new column is an AUTOMATIC INSERT AS or an AUTOMATIC AS (implying both insert and update actions) then all previously inserted rows are updated with the AUTOMATIC expression as the value of the new column.

- SQL automatically propagates to all referencing view definitions any domain or data type attributes provided by an ALTER COLUMN clause.

  For example, if you modify a column to refer to a domain for its data type, any view column which refers to that column directly is updated to reflect the new attributes of the modified column.

```
SQL> CREATE DOMAIN d CHAR(1);
SQL> CREATE TABLE T (A d);
SQL> CREATE VIEW V (B,C) AS SELECT A, A||'X' FROM T;
SQL> SHOW VIEW (COL) V;
Information for table V

Columns for view V:
Column Name                    Data Type        Domain
-----------                    ---------        ------
B                              CHAR(1)
C                              CHAR(2)
SQL> ALTER DOMAIN d CHAR(5);
SQL> SHOW VIEW (COL) v;
Information for table V

Columns for view V:
Column Name                    Data Type        Domain
-----------                    ---------        ------
B                              CHAR(5)
C                              CHAR(6)
```

- Any table referenced by a COMPUTED BY, AUTOMATIC or DEFAULT clause will be implicitly reserved for SHARED READ by Oracle Rdb when the column is referenced in a query. Therefore, it is not necessary to explicitly reserve these tables in the DECLARE TRANSACTION or SET TRANSACTION statement unless the required lock mode is higher than SHARED READ.

  If any of these expressions call an SQL function which reads from a table or view, then these tables are not implicitly reserved. You must include a LOCK TABLE statement in the function (or any called procedure) to ensure that references to the tables are allowed, even when not listed in the DECLARE TRANSACTION or SET TRANSACTION statement.

- When an ALTER TABLE statement drops one or more LIST OF BYTE VARYING columns, the ALTER TABLE statement must read each row in the table and record the pointers for the LIST values. This list is processed at COMMIT time to delete the LIST column data. Therefore, the database administrator must also allow for this time when altering the table.

  Reserving the table for EXCLUSIVE WRITE is recommended because the dropped LIST columns will require that each row in the table be updated and set the LIST column to NULL - it is this action which queues the pointers for commit time processing. This reserving mode will eliminate snapshot file I/O, lower lock resources and reduce virtual memory usage.

  As the LIST data is stored outside the table, performance may be improved by attaching to the database with the RESTRICTED ACCESS clause, which has the side effect of reserving all the LIST storage areas for EXCLUSIVE access, and therefore eliminates snapshot I/O during the delete of the LIST data.

- The sequence created by the IDENTITY attribute can be shown with SHOW SEQUENCES, and the attributes of the sequence can be altered using ALTER SEQUENCE, COMMENT ON SEQUENCE, GRANT and REVOKE.

  However, neither DROP SEQUENCE or RENAME SEQUENCE are permitted for this special sequence. A DROP TABLE, or an ALTER TABLE . . . DROP COLUMN of the identity column will implicitly drop the identity sequence. A RENAME of the table will implicitly rename the matching identity sequence.

- Constraints and indices may be created for the identity column. Indices can improve query performance, and constraints such as PRIMARY KEY or UNIQUE will allow references from other tables FOREIGN KEY constraints.

- The ALTER TABLE statement can reference a table reserved in DATA DEFINITION mode.

  The following clauses are supported:

  - ADD CONSTRAINT
  - ALTER COLUMN...CONSTRAINT
  - ENABLE CONSTRAINT, ENABLE PRIMARY KEY, and ENABLE UNIQUE (...)
  - DROP CONSTRAINT
  - ALTER COLUMN ... NULL

  - DISABLE CONSTRAINT, DISABLE PRIMARY KEY, and DISABLE UNIQUE
  - DISABLE UNIQUE (...)

  The ADD and ENABLE CONSTRAINT clauses are best suited to concurrent execution as they may require I/O to validate the constraint.

- Most ALTER TABLE clauses are supported for tables reserved for SHARED DATA DEFINITION. The exceptions are those clauses that change the structure of the table: ADD COLUMN and DROP COLUMN, and ALTER COLUMN which changes the data type.
- NOT NULL constraints, and the not null restriction for PRIMARY KEY constraints are validated using a single table scan for all such constraints on a table. This should reduce the table sequential scans in the absence of a suitable index for each constraint. Note that the side effect is that only the first failing constraint name is reported.
- When multiple tables are verified, the constraints are ordered by table name. The goal is to make use of any buffered table rows for subsequent constraint queries. In prior versions, the constraints were verified in (approximate) definition order which might result in other tables being read and buffered data not being available.
- During RMU Verify Constraints, or during ALTER TABLE ... ENABLE ALL CONSTRAINTS, more detailed information on the verify can be seen by defining the ITEM_LIST flag. This can be done using either the SET FLAGS statement in SQL or defining the logical name RDMS$SET_FLAGS.

  In the case of a failing NOT NULL constraint, the DBKEY of the failing row is reported. The failure status (VMS condition code) is also reported along with the associated message text.

Examples

Example 1: Adding a column to the EMPLOYEES table

```
SQL> ALTER TABLE EMPLOYEES ADD SALARY INTEGER(2);
```

Example 2: Adding a column and altering a column in the COLLEGES table

The following example adds two columns, one with a query name to the COLLEGES table. ALTER DOMAIN is also used to implicitly alter the POSTAL_CODE column to accept 9 characters instead of 5.

```
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 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> create table DAILY_SALES
cont>     (hours_overtime     int
cont>     ,hours_worked       int default 0
cont>     ,gross_sales        int
cont>     ,salesperson        char(20)
cont>     );
SQL>
SQL> /* Change the default value for the column HOURS_OVERTIME
***> */
SQL> alter table DAILY_SALES
cont>     alter column HOURS_OVERTIME
cont>   set default 0;
SQL>
SQL> /* Insert the days sales figures into the table,
***>    accepting the default values for HOURS_WORKED, and
***>    HOURS_OVERTIME
***> */
SQL> insert into DAILY_SALES (gross_sales, salesperson)
cont>     values (2567, 'Bartlett');
1 row inserted
SQL>
SQL> table DAILY_SALES;
 HOURS_OVERTIME   HOURS_WORKED   GROSS_SALES   SALESPERSON
              0              0          2567   Bartlett
1 row selected
SQL>
```

Example 4: Deleting a constraint from the EMPLOYEES table

To find out the name of a constraint, use the SHOW TABLES statement. The SHOW TABLES statement shows all constraints that refer to a table, not just those defined as part of the table's definition. For that reason it is good practice to always use a prefix to identify the table associated with a constraint when you assign constraint names with the CONSTRAINT clause.

The constraint DEGREES_FOREIGN1 in this SHOW display follows that convention to indicate that the constraint is associated with the DEGREES, not the EMPLOYEES, table despite the constraint's 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;
```

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>
```

Example 8: Using the Position Clause

```
SQL> SHOW TABLE (COL) EMPLOYEES
Information for table EMPLOYEES
Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          ID_NUMBER
 Missing Value:
LAST_NAME                      CHAR(14)         LAST_NAME
FIRST_NAME                     CHAR(10)         FIRST_NAME
MIDDLE_INITIAL                 CHAR(1)          MIDDLE_INITIAL
 Missing Value:
ADDRESS_DATA_1                 CHAR(25)         ADDRESS_DATA_1
 Missing Value:
ADDRESS_DATA_2                 CHAR(25)         ADDRESS_DATA_2
 Missing Value:
CITY                           CHAR(20)         CITY
 Missing Value:
STATE                          CHAR(2)          STATE
 Missing Value:
POSTAL_CODE                    CHAR(5)          POSTAL_CODE
 Missing Value:
SEX                            CHAR(1)          SEX
 Missing Value: ?
BIRTHDAY                       DATE VMS         STANDARD_DATE
 Missing Value: 17-NOV-1858 00:00:00.00
STATUS_CODE                    CHAR(1)          STATUS_CODE
 Missing Value: N
```

```
SQL> -- Alter the table to rearrange the order in which columns
SQL> -- are displayed.
SQL> ALTER TABLE EMPLOYEES
cont> ALTER COLUMN SEX BEFORE COLUMN LAST_NAME
cont> ALTER COLUMN BIRTHDAY BEFORE COLUMN LAST_NAME
cont> ALTER COLUMN STATUS_CODE BEFORE COLUMN LAST_NAME;
SQL> COMMIT;
SQL> -- Show the table to demonstrate that the order in which
SQL> -- columns are displayed has changed.
SQL> SHOW TABLE (COL) EMPLOYEES;
Information for table EMPLOYEES
Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          ID_NUMBER
 Missing Value:
SEX                            CHAR(1)          SEX
 Missing Value: ?
BIRTHDAY                       DATE VMS         STANDARD_DATE
 Missing Value: 17-NOV-1858 00:00:00.00
STATUS_CODE                    CHAR(1)          STATUS_CODE
 Missing Value: N
LAST_NAME                      CHAR(14)         LAST_NAME
FIRST_NAME                     CHAR(10)         FIRST_NAME
MIDDLE_INITIAL                 CHAR(1)          MIDDLE_INITIAL
 Missing Value:
ADDRESS_DATA_1                 CHAR(25)         ADDRESS_DATA_1
 Missing Value:
ADDRESS_DATA_2                 CHAR(25)         ADDRESS_DATA_2
 Missing Value:
CITY                           CHAR(20)         CITY
 Missing Value:
STATE                          CHAR(2)          STATE
 Missing Value:
POSTAL_CODE                    CHAR(5)          POSTAL_CODE
 Missing Value:
```

Example 9: Disabling a Trigger

```
SQL> SELECT * FROM JOB_HISTORY WHERE 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> DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID ='00164';
1 row deleted
SQL> -- Show that the EMPLOYEE_ID_CASCADE_DELETE trigger caused
SQL> -- records in the JOB_HISTORY table to be deleted for the
SQL> -- employee with EMPLOYEE_ID of 00164.
SQL> SELECT * FROM JOB_HISTORY WHERE EMPLOYEE_ID='00164';
0 rows selected
SQL> -- Roll back the delete operation and alter the EMPLOYEES table
SQL> -- to disable the EMPLOYEE_ID_CASCADE_DELETE trigger.
SQL> ROLLBACK;
SQL> ALTER TABLE EMPLOYEES
cont> DISABLE TRIGGER EMPLOYEE_ID_CASCADE_DELETE;
SQL> -- Commit the alter operation and disconnect to ensure that
SQL> -- the next connection will have the trigger disabled.
SQL> COMMIT;
SQL> DISCONNECT DEFAULT;
SQL> ATTACH 'FILENAME MF_PERSONNEL.RDB';
SQL> DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID ='00164';
1 row deleted
SQL> -- Show that with the trigger disabled, a deletion of
SQL> -- employee 00164 from the EMPLOYEES table does not
SQL> -- trigger a deletion for that employee from the
SQL> -- JOB_HISTORY table.
SQL> SELECT * FROM JOB_HISTORY WHERE 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
```

Example 10: NOT NULL constraint is dropped

The following example shows that the NOT NULL constraint is dropped by ALTER TABLE.

```
SQL> create table MY_TABLE (a integer not null);
SQL>
SQL> show table (constraint) MY_TABLE
Information for table MY_TABLE

Table constraints for MY_TABLE:
MY_TABLE_A_NOT_NULL
 Not Null constraint
 Column constraint for MY_TABLE.A
 Evaluated on UPDATE, NOT DEFERRABLE
 Source:
        MY_TABLE.A NOT null

Constraints referencing table MY_TABLE:
No constraints found

SQL>
SQL> alter table MY_TABLE
cont>     alter column A NULL;
SQL>
SQL> show table (constraint) MY_TABLE
Information for table MY_TABLE

Table constraints for MY_TABLE:
No constraints found

Constraints referencing table MY_TABLE:
No constraints found

SQL>
```

Example 11: Adding an identity column to an existing table

```
SQL> alter table EMPLOYEES
cont>   add column SEQUENCE_ID integer identity (1000, 10)
cont>   comment is 'Add unique sequence number for every employee';
SQL>
SQL> show table (column) EMPLOYEES
Information for table EMPLOYEES

Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          ID_NUMBER
.
.
.
SEQUENCE_ID                    INTEGER
 Computed:      IDENTITY
 Comment:       Add unique sequence number for every employee
```

```
SQL> select EMPLOYEE_ID, SEQUENCE_ID from employees;
 EMPLOYEE_ID   SEQUENCE_ID
 00164                1000
 00165                1010
.
.
.
 00418                1970
 00435                1980
 00471                1990
100 rows selected
SQL>
SQL> show sequence EMPLOYEES
     EMPLOYEES
 Sequence Id: 2
 Initial Value: 1000
 Minimum Value: 1000
 Maximum Value: (none)
 Next Sequence Value: 2000
 Increment by: 10
 Cache Size: 20
 No Order
 No Cycle
 No Randomize
 Wait
 Comment:       column IDENTITY sequence
SQL>
```

Example 12: Revising a COMPUTED BY column

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

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

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

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

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

Example 13: Using the COMPILE ALL TRIGGERS option

```
SQL> set flags 'warn_invalid';
SQL>
SQL> alter module M
cont>     drop procedure PP cascade;
~Xw: Trigger "C_INSERT" marked invalid
SQL>
SQL> show trigger C_INSERT
     C_INSERT
 Current state is INVALID
    Can be revalidated
 Source:
 c_insert
    after insert on C
    when (C.b is NULL)
        (call PP())
    for each row
SQL>
SQL> alter table C
cont>     compile all triggers;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-OBSOLETE_METADA, request references metadata objects that no longer exist
-RDMS-F-BAD_SYM, unknown routine symbol - PP
SQL>
SQL> ! Replace missing procedure PP
SQL> alter module M
cont>     add procedure PP;
```

```
cont>     trace 'in PP';
cont> end module;
SQL>
SQL> alter table C
cont>     compile all triggers;
SQL>
SQL> ! Show that the INVALID flag is now cleared
SQL> show trigger C_INSERT
     C_INSERT
 Source:
 c_insert
    after insert on C
    when (C.b is NULL)
       (call PP())
    for each row
SQL>
```

ALTER TRIGGER Statement

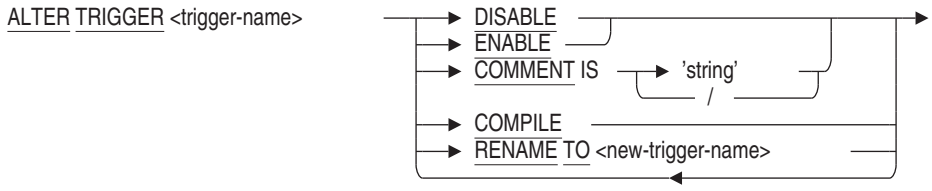
Enables, disables, or renames an existing trigger. Changes take place after the transaction containing the ALTER TRIGGER statement is committed.

Environment

You can use the ALTER 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



Arguments

COMMENT IS 'string'
Adds a comment about the trigger. SQL displays the text of the comment when it executes a SHOW statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

COMPILE
This clause directs Rdb to re-compile the trigger to ensure that it is valid. If COMPILE is successful and the trigger was marked invalid, but "Can be revalidated", then the invalid flag will be cleared.

Triggers can be marked "invalid" when a procedure, function or sequence is droped using the CASCADE clause.

DISABLE
Disables a previously enabled trigger.

ENABLE
Enables a previously disabled trigger.

RENAME TO
Changes the name of the trigger being altered. See the RENAME Statement for further discussion. If the new name is the name of a synonym then an error will be raised.

The RENAME TO clause requires synonyms be enabled for this database. Refer to the ALTER DATABASE Statement SYNONYMS ARE ENABLED clause. Note that these synonyms may be deleted if they are no longer used by database definitions or applications.

trigger-name
The name of an existing trigger.

Usage Notes

- The user must have ALTER privilege on the triggering table.
- The user must also have DROP privilege on the table to use an ALTER TRIGGER DISABLE statement (if the trigger is currently enabled), and the user must also have the CREATE privilege on the table to use an ALTER TRIGGER ENABLE statement (if the trigger is currently disabled).
- By default, a trigger is enabled when it is created.
- When a trigger is disabled, it is not executed by the INSERT, UPDATE, or DELETE statement.
- When a previously disabled trigger is reenabled, it takes effect for new queries only. Existing cursors that have been opened do not see the enabled triggers. Therefore, it is recommended that when you enable or disable a trigger, you commit, disconnect, and reattach so that consistent behavior is seen by all queries.
- Use the SHOW TRIGGERS statement to display the setting (disabled or enabled) for a trigger.
- The COMMENT IS clause is equivalent to the COMMENT ON TRIGGER statement.
- Any trigger marked with the invalid flag will still be used by Oracle Rdb if at runtime it can be compiled successfully. However, only the COMPILE clause of the ALTER TRIGGER statement, or the COMPILE

ALL TRIGGERS clause of the ALTER TABLE statement will clear the "invalid" flag.

Examples

Example 1: Disabling a Trigger

The following example shows that while the EMPLOYEE_ID_CASCADE_ DELETE trigger is enabled, deleting a record from EMPLOYEES causes the corresponding record in JOB_HISTORY to be deleted. After the trigger is disabled, a deletion from EMPLOYEES does not trigger a deletion from the JOB_HISTORY table.

```
SQL> SELECT * FROM JOB_HISTORY WHERE 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> DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID ='00164';
1 row deleted
SQL> SELECT * FROM JOB_HISTORY WHERE EMPLOYEE_ID='00164';
0 rows selected
SQL> ROLLBACK;
SQL> ALTER TRIGGER EMPLOYEE_ID_CASCADE_DELETE DISABLE;
SQL> COMMIT;
SQL> DISCONNECT DEFAULT;
.
.
.
SQL> DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID ='00164';
1 row deleted
SQL> SELECT * FROM JOB_HISTORY WHERE 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
```

Example 2: Using the COMPILE option

```
SQL> set flags 'warn_invalid';
SQL>
SQL> alter module M
cont>     drop procedure PP cascade;
~Xw: Trigger "C_INSERT" marked invalid
SQL>
SQL> show trigger C_INSERT
     C_INSERT
 Current state is INVALID
    Can be revalidated
 Source:
 c_insert
    after insert on C
    when (C.b is NULL)
        (call PP())
    for each row
SQL>
SQL>
SQL> alter trigger C_INSERT
cont>     compile;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-OBSOLETE_METADA, request references metadata objects that no longer exist
-RDMS-F-BAD_SYM, unknown routine symbol - PP
SQL>
SQL> alter module M
cont>     add procedure PP;
cont>     trace 'in PP';
cont> end module;
SQL>
SQL> alter trigger C_INSERT
cont>     compile;
SQL>
SQL> show trigger C_INSERT
     C_INSERT
 Source:
 c_insert
    after insert on C
    when (C.b is NULL)
        (call PP())
    for each row
SQL>
```

ALTER USER Statement

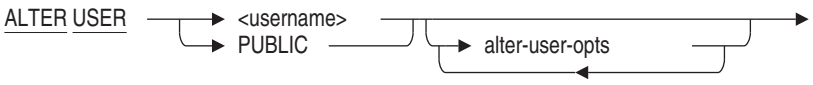
Modifies an entry for the specified user name. The modifications take effect on the next database connection after the ALTER USER statement is committed.

Environment

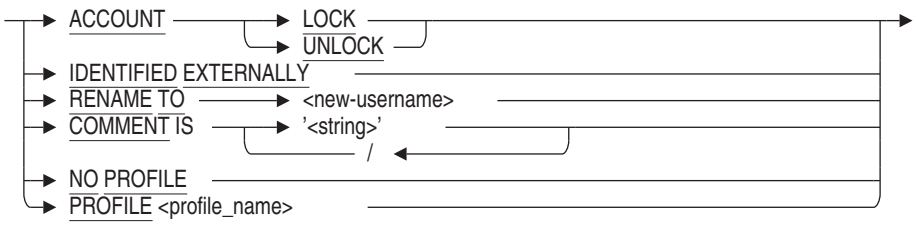
You can use the ALTER USER statement:

- 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-user-opts =



Arguments

ACCOUNT LOCK
ACCOUNT UNLOCK
The ACCOUNT LOCK clause disables access to the database by the user for whom the ALTER USER statement is being applied. The ACCOUNT UNLOCK clause allows the user access to the database.

COMMENT IS 'string'
Adds a comment about the user. SQL displays the text of the comment when it executes a SHOW USERS statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

IDENTIFIED EXTERNALLY
Indicates that the user will be authenticated through the operating system.

NO PROFILE
Removes any assigned profile from the identified user. No error is returned if a profile is not currently assigned.

PROFILE
Identifies a new profile for assignment to the user and replaces any previously assigned profile. The specified profile name must be the name of an existing profile.

PUBLIC
The PUBLIC user in the database. This entry gives you control over anonymous users who access the database.

RENAME TO new-username
Changes the user name and, if a security profile exists, assigns the security profile associated with the old user name to the new user name. This might be used, for example, when a person's name changes (as through marriage), and, therefore, his or her account on the operating system is changed accordingly. The new-username must not currently exist in the database.

When the ALTER USER command is issued, the existing user name is removed from the database and replaced with the new-username. If SECURITY CHECKING is INTERNAL, then subsequent SHOW PROTECTION statements will display the new name for the user, and all GRANT and REVOKE statements will require the new-username. The new-username is not visible to other sessions until the transaction containing the ALTER USER command is committed.

See the RENAME Statement for further discussion.

username
An existing user name in the database.

Usage Notes

- You must have the SECURITY privilege on the database, or the OpenVMS SECURITY privilege, to alter a user.
- You can display existing users defined for a database by issuing a SHOW USERS statement.
- When you issue the RENAME clause, the new user name must exist as an operating system user name. See the Examples section.

Examples

Example 1: Renaming a User

```
SQL> create user KELLYN
cont>   identified externally
cont>   comment is 'User: Edward "Ned" Kelly';
SQL>
SQL> -- The alternate name must exists at the operating system level
SQL> alter user KELLYN rename to N_KELLY;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-E-NOSUCHPRF, unknown profile user or role
SQL>
SQL> -- Use the new corporate user-id naming scheme
SQL> alter user KELLYN
cont>   rename to NKELLY;
```

Example 2: Adding a profile to a user

This example creates a new profile that defines the DEFAULT transaction and then assigns a profile to the user. The next time the user attaches to the database, the START DEFAULT TRANSACTION statement will use the defined profile instead of the standard READ ONLY default.

```
SQL> create profile READ_COMMITTED
cont> default transaction read write isolation level read committed wait 30;
SQL> show profile READ_COMMITTED
      READ_COMMITTED
      Default transaction read write wait 30
        Isolation level read committed
SQL> alter user JAIN profile READ_COMMITTED;
SQL> show user JAIN;
      JAIN
      Identified externally
      Account is unlocked
      Profile: READ_COMMITTED
      No roles have been granted to this user
```

ALTER VIEW Statement

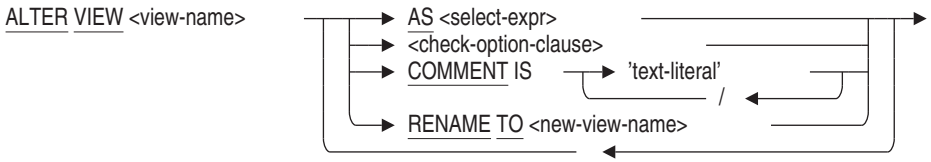
This statement allows the named view to be modified.

Environment

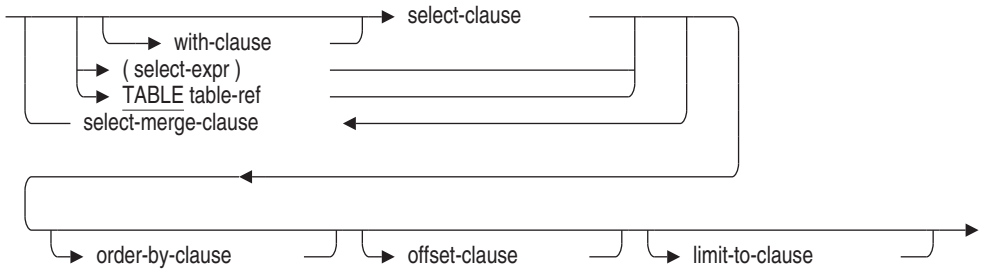
You can use the ALTER VIEW statement:

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

Format



select-expr =



check-option-clause =

Arguments

AS
Replaces the view select expression and the definitions of the columns. The number of expressions in the select list must match the original CREATE VIEW column list.

COMMENT IS
Replaces the comment currently defined for the view (if any). The comment will be displayed by the SHOW VIEW statement in Interactive SQL.

CONSTRAINT check-option-name
Specifies 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. 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.

RENAME TO
Renames the current view. The new view name must not exist as the name of an existing view, table, sequence, or synonym.

WITH CHECK OPTION
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.)

WITH NO CHECK OPTION
Removes any check option constraint currently defined for the view.

Usage Notes

- You must have ALTER privilege on the referenced view.
- The ALTER VIEW statement causes the RDB$LAST_ALTERED column of the RDB$RELATIONS table for the named view to be updated with the transaction's timestamp.

- Neither the column names nor their position may be modified using the ALTER VIEW statement, nor can columns be added or dropped for a view. These changes require both DROP VIEW and CREATE VIEW statements to replace the existing definition.
- The RENAME TO clause allows the name of the view to be changed. This clause requires that synonyms are enabled in the database. To enable synonyms, use the ALTER DATABASE ...SYNONYMS ARE ENABLED statement.

  The old name will be used to create a synonym for the new name of this view. This synonym can be dropped if the name is no longer used by database definitions or applications.

  This clause is equivalent to the RENAME VIEW statement.
- The COMMENT IS clause changes the existing comment on the view. This clause is equivalent to the COMMENT ON VIEW statement.
- Changes to the column expression may change the column to read-only and prevent referencing routines, triggers, and applications from performing INSERT and UPDATE operations on those columns. Such changes are reported at run time.

  Similarly, if the view select table expression becomes read-only, referencing queries may fail.

  SQL considers as read-only views those with select expressions that:

  - Use the DISTINCT clause to eliminate duplicate rows from the result table
  - Name more than one table or view in the FROM clause
  - Use a derived table in the FROM clause
  - Include a statistical function in the select list
  - Contain a UNION, EXCEPT DISTINCT (MINUS), INTERSECT DISTINCT, GROUP BY, or HAVING clause
- If the AS clause changes the view to read-only, or includes a LIMIT TO ... ROWS clause on the main query, then the check option constraint is implicitly removed.
- The ALTER VIEW statement can reference a table reserved in DATA DEFINITION mode.

Examples

Example 1: Changing the comment on a view

A comment can be added or changed on a view using the COMMENT IS clause as shown in this example.

```
SQL> show view (comment) current_job
Information for table CURRENT_JOB

SQL> alter view CURRENT_JOB
cont>   comment is 'Select the most recent job for the employee';
SQL> show view (comment) current_job
Information for table CURRENT_JOB

Comment on table CURRENT_JOB:
Select the most recent job for the employee

SQL>
```

Example 2: Changing the column's results of a view definition

The following view uses a derived table and join to collect the count of employees in each department. The view is used in several reporting programs used by the department and company managers.

```
SQL> create view DEPARTMENTS_SUMMARY
cont> as
cont> select department_code, d.department_name,
cont>        d.manager_id, jh.employee_count
cont> from departments d inner join
cont>      (select department_code, count (*)
cont>         from job_history
cont>         where job_end is null
cont>         group by department_code)
cont>           as jh (department_code, employee_count)
cont>      using (department_code);
SQL>
SQL> show view DEPARTMENTS_SUMMARY;
Information for table DEPARTMENTS_SUMMARY
```

```
Columns for view DEPARTMENTS_SUMMARY:
Column Name                    Data Type        Domain
-----------                    ---------        ------
DEPARTMENT_CODE                CHAR(4)
DEPARTMENT_NAME                CHAR(30)
 Missing Value: None
MANAGER_ID                     CHAR(5)
 Missing Value:
EMPLOYEE_COUNT                 INTEGER
 Source:
select department_code, d.department_name,
       d.manager_id, jh.employee_count
from departments d inner join
     (select department_code, count (*)
        from job_history
        where job_end is null
        group by department_code) as jh (department_code, employee_count)
     using (department_code)

SQL>
```

The database administrator decides to create a column in the DEPARTMENTS table to hold the count of employees (rather than using a query to gather the total) and to maintain the value through triggers on EMPLOYEES and JOB_HISTORY (not shown here). Now the view can be simplified without resorting to a DROP VIEW and CREATE VIEW. The ALTER VIEW statement preserves the dependencies on the view from other views, triggers, and routines and so minimizes the work required to implement such a change.

```
SQL> alter table DEPARTMENTS
cont>     add column EMPLOYEE_COUNT integer;
SQL>
SQL> alter view DEPARTMENTS_SUMMARY
cont> as
cont> select department_code, d.department_name,
cont>        d.manager_id, d.employee_count
cont> from departments d;
SQL>
SQL> show view DEPARTMENTS_SUMMARY;
Information for table DEPARTMENTS_SUMMARY
```

```
Columns for view DEPARTMENTS_SUMMARY:
Column Name                    Data Type        Domain
-----------                    ---------        ------
DEPARTMENT_CODE                CHAR(4)
 Missing Value: None
DEPARTMENT_NAME                CHAR(30)
 Missing Value: None
MANAGER_ID                     CHAR(5)
 Missing Value:
EMPLOYEE_COUNT                 INTEGER
 Source:
select department_code, d.department_name,
       d.manager_id, d.employee_count
from departments d

SQL>
```

Example 3: Changing the WITH CHECK OPTION constraint of a view definition

This example shows that a WITH CHECK OPTION constraint restricts the inserted data to the view's WHERE clause. Once the constraint is removed, the INSERT is no longer constrained.

```
SQL> create view TOLIVER_EMPLOYEE
cont> as select * from EMPLOYEES where employee_id = '00164'
cont> with check option;
SQL> insert into TOLIVER_EMPLOYEE (employee_id) value ('00000');
%RDB-E-INTEG_FAIL, violation of constraint TOLIVER_EMPLOYEE_CHECKOPT1 caused operation to fail
-RDB-F-ON_DB, on database DISK1:[DATABASES]MF_PERSONNEL.RDB;1
SQL>
SQL> alter view TOLIVER_EMPLOYEE with no check option;
SQL>
SQL> insert into TOLIVER_EMPLOYEE (employee_id) value ('00000');
1 row inserted
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.

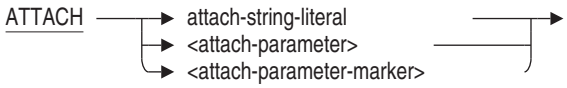
If a database is currently attached and you attach to another database without using an alias, SQL detaches the current database environment and attaches to the specified one in its place.

Environment

You can use the ATTACH statement:

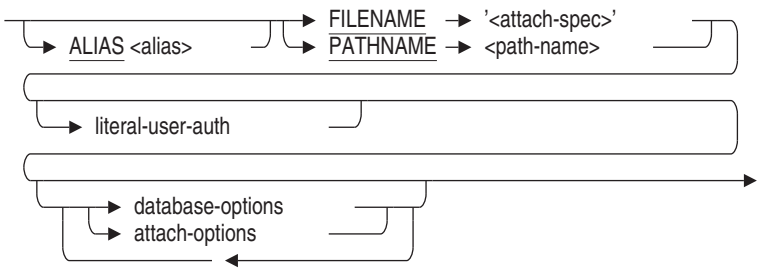
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

Format

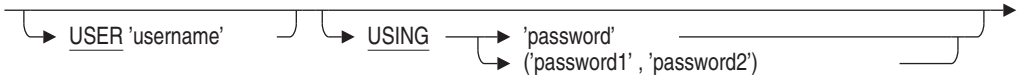


attach-string-literal =

attach-expression =

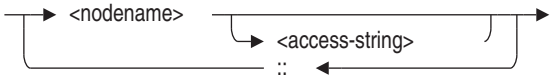


literal-user-auth =

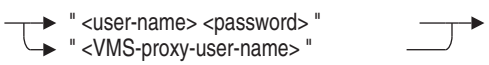


attach-spec =

node-spec =

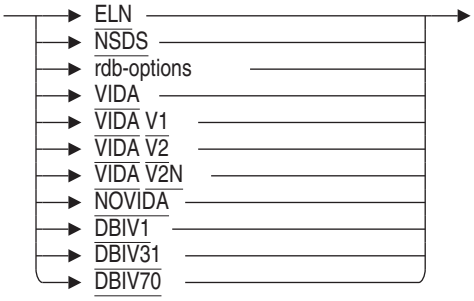


access-string =

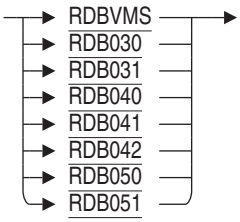


ATTACH Statement

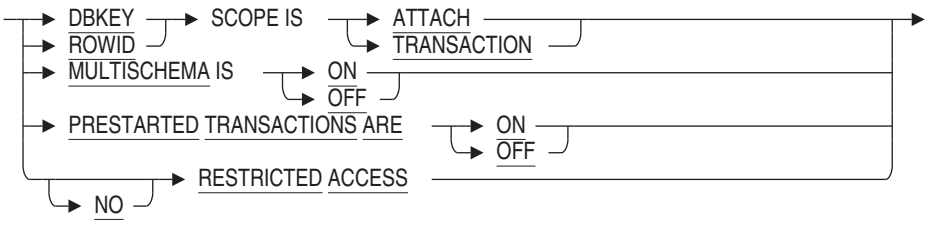
database-options =



rdb-options =



attach-options =



Arguments

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
```

attach-expression
Specifies a database to be added to the environment.

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-options
Specifies characteristics of the particular database attach. You can specify more than one of these clauses.

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.

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.

DBKEY SCOPE IS ATTACH
DBKEY SCOPE IS TRANSACTION
Controls when the database key of a deleted row can be used again by SQL.

- 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.

- The DBKEY SCOPE IS ATTACH clause means that SQL cannot use the database key again (to refer to a newly inserted row) until all users who have attached with DBKEY SCOPE IS ATTACH have detached from the database.

  It only requires one process to attach with DBKEY SCOPE IS ATTACH to force all database users to assume this characteristic.

- Oracle Corporation recommends using DBKEY SCOPE IS TRANSACTION to prevent excessive consumption of storage area space by overhead space needed to support DBKEY SCOPE IS ATTACH, and to prevent performance problems when storing new rows.

  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.

For more information, see Section 2.6.5.

DISPLAY CHARACTER SET support-char-set
Specifies the character set encoding and characteristics expected of text strings returned back to SQL from Oracle Rdb. See the Usage Notes under CREATE DATABASE Statement for additional information.

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 2.2.8.1.

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.

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, 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 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
```

- Use a DECnet 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 Rdb Guide to SQL Programming*.

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
```

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.

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 to define the default setting for prestarted transactions outside of an application. The PRESTARTED TRANSACTION clause overrides this logical name. For more information, see the *Oracle Rdb7 Guide to Database Performance and Tuning*. See also the ALTER and CREATE DATABASE clause PRESTARTED TRANSACTIONS ARE ENABLED for more details.

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.

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.

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'
USING ('password', 'password')
A character string literal that specifies the user's password for the user name specified in the USER clause. If the user requires two passwords, then specify both values in parenthesis and separated by a comma.

Usage Notes

- 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';
```

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.

```
SQL> ATTACH 'FILENAME FARSID::USER1:[GREMBOWSKI.DB]MF_PERSONNEL -
cont>        USER ''grembowski'' USING ''mypassword''';
```

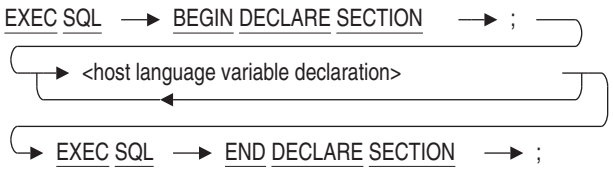
BEGIN DECLARE Statement

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.

END DECLARE SECTION
Delimits the end 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 2.2.13 for full details on host language variable definitions.

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.

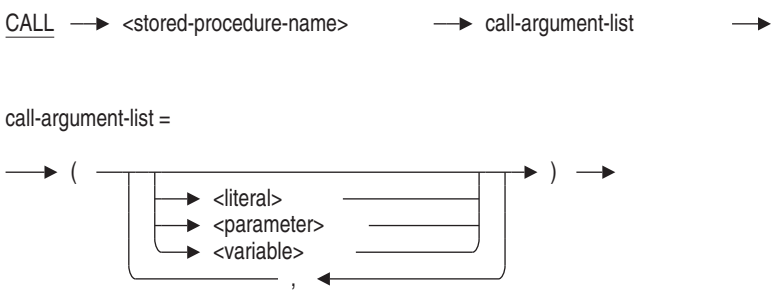
For optional information on invoking stored procedures, see the CALL Statement for Compound Statements.

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 that are dynamically executed

Format



Arguments

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 2.2.13 for details about how to use parameters in programs for static as well as dynamic SQL statement execution.

procedure-name
The name of a stored procedure.

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.

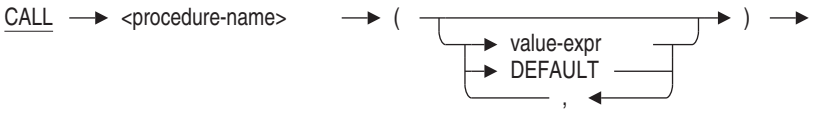
For optional information on invoking stored procedures, see the CALL Statement for Simple Statements.

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

DEFAULT
Requests that Oracle Rdb use the DEFAULT expression defined for the parameter. The DEFAULT can be defined for an IN parameter during the CREATE MODULE . . . PROCEDURE or ALTER MODULE . . . ADD

PROCEDURE statements. If no DEFAULT clause exists then the NULL expression is assumed.

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.
- Oracle Rdb allows the CALL statement in a compound statement to omit trailing IN mode parameters which have had a DEFAULT value defined in the procedure definition. Also supported is the DEFAULT keyword to replace an explicit value for the parameter.

However, the simple CALL statement (used outside a BEGIN END block) is not adaptable in this way and requires a full set of parameters and values. This is because a parameter signature is calculated for this type of CALL statement so that the parameter block passed by the calling routine and used by the called routine match exactly in parameter count and data types.

This is a permanent restriction for the simple CALL statement.

The following example shows that truncated parameter lists are fully supported by the compound use form of the CALL statement, but not by the simple CALL statement.

```
SQL> ATTACH 'FILENAME db$:scratch';
SQL> CREATE MODULE mmm
cont> PROCEDURE mmm_p (IN :a INTEGER DEFAULT 0, IN :b INTEGER DEFAULT 1);
cont> TRACE :a, :b;
cont> END MODULE;
SQL> SET FLAGS 'Trace';
SQL> CALL mmm_p (10,20);
~Xt: 10         20
SQL> CALL mmm_p (10);
%SQL-F-ARGCOUNT, Procedure MMM_P expected 2 parameters, was passed 1
SQL> call MMM_P ();
%SQL-F-ARGCOUNT, Procedure MMM_P expected 2 parameters, was passed 0
SQL> begin
cont> CALL mmm_p (10,20);
cont> CALL mmm_p (10);
cont> call mmm_p ();
cont> END;
~Xt: 10         20
~Xt: 10         1
~Xt: 0          1
```

For maximum flexibility, use the CALL statement inside a compound statement which supports truncated parameter lists, the DEFAULT keyword, and full value expressions for parameter arguments.

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 (Searched) Control Statement

Executes one of a sequence of alternate statement blocks in a compound statement of a multistatement procedure. Unlike the simple CASE control statement, the searched CASE control statement supports arbitrary predicates for the WHEN clause that can contain variable and parameter references.

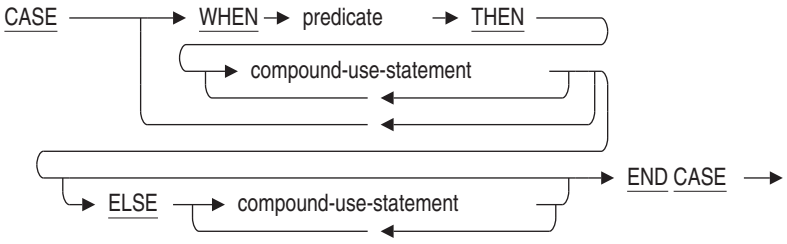
Environment

You can use the searched 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

case-searched-statement =



Arguments

ELSE compound-use-statement
Specifies the set of SQL statements to be executed when the WHEN clause evaluates to FALSE or UNKNOWN.

THEN compound-use-statement
Specifies the set of SQL statements to be executed when the WHEN clause evaluates to TRUE.

WHEN predicate
Determines whether the compound use statements in the THEN clause are to be executed or the compound use statements in the ELSE clause are to be executed. If the predicate evaluates to TRUE, then the compound use statements in the THEN clause are executed. If the predicate evaluates to FALSE or UNKNOWN, then the compound use statements in the ELSE clause are executed.

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.

Examples

Example 1: Specifying Predicates with Variable References

```
SQL> CREATE TABLE T (C INT);
SQL> BEGIN
cont> DECLARE :V INTEGER = 10;
cont> DECLARE :X INTEGER = 0;
cont> CASE
cont>   WHEN :V = 1 THEN INSERT INTO T(C) VALUES (:X + 1);
cont>   WHEN :V = 2 THEN INSERT INTO T(C) VALUES (:X + 2);
cont>   WHEN :V > 3 THEN INSERT INTO T(C) VALUES (:X);
cont>   ELSE INSERT INTO T(C) VALUES (-1);
cont> END CASE;
cont> END;
```

CASE (Simple) Control Statement

Executes one of a sequence of alternate statement blocks in a compound statement of a multistatement procedure.

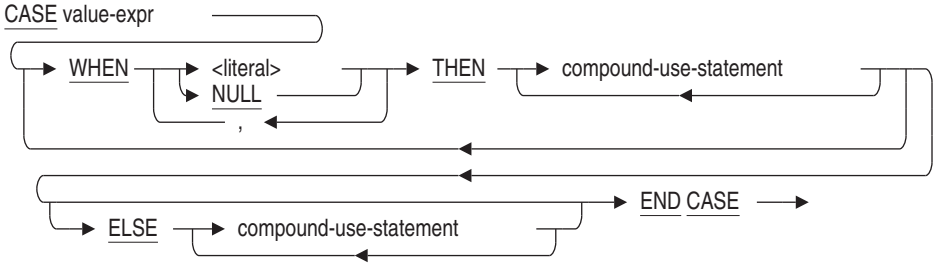
Environment

You can use the simple 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

case-statement =



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.

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.

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.

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.

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. The simple CASE statement allows you to provide a list of literal values for every WHEN clause.

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 ;
```

Example 2: Using a List of Literal Values with the Case Statement

```
SQL> DECLARE :CODE CHAR(4);
SQL> BEGIN
cont>    JOB_LOOP:
cont>        FOR :JOBFOR
cont>          AS EACH ROW OF
cont>          SELECT * FROM JOBS JOB
cont>          DO
cont>          SET :CODE = :jobfor.JOB_CODE;
cont>            CASE :CODE
cont>              WHEN 'ASCK' THEN
cont>                 UPDATE JOBS
cont>                 SET MINIMUM_SALARY=10000
cont>                 WHERE JOB_CODE = :code;
cont>              WHEN 'ADMN', 'JNTR', 'SCTR' THEN
cont>                 UPDATE JOBS
cont>                 SET MINIMUM_SALARY=15000
cont>                 WHERE JOB_CODE = :code;
cont>              ELSE
cont>                 UPDATE JOBS
cont>                 SET MINIMUM_SALARY=:jobfor.MINIMUM_SALARY*1.1
cont>                 WHERE JOB_CODE=:code;
cont>            END CASE;
cont>        END FOR;
cont> END;
SQL>
```

CLOSE Statement

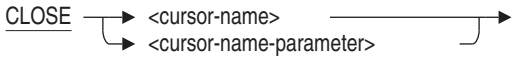
Closes an open cursor.

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

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 the following database objects:

- Catalog
- Collating sequence
- Column
- Constraint
- Domain
- Database
- Function
- Index
- Index partition
- Module
- Outline
- Procedure
- Profile
- Role
- Schema
- Sequence
- Storage map
- Storage map partition
- Synonym
- Table
- Trigger
- User
- View

SQL displays the comments on these objects when you issue a SHOW statement.

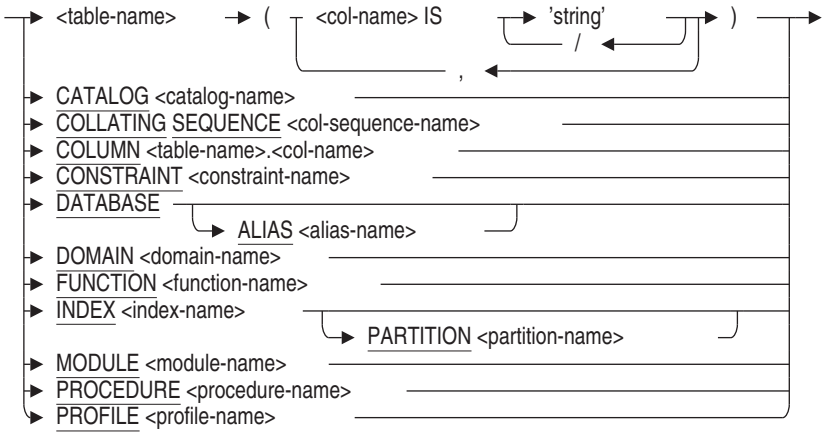
Environment

You can use the COMMENT ON statement:

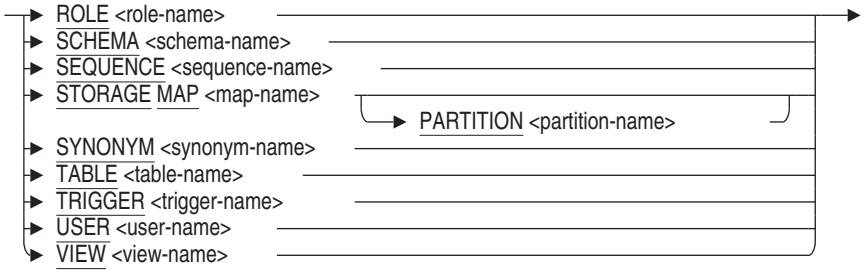
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

Format

objects1 =



objects2 =



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 alias name.

COLLATING SEQUENCE column-sequence-name
Names the collating sequence for which you want to create a comment. If the collating sequence is not in the default schema, you must qualify the collating sequence name in the COMMENT ON statement with an alias name.

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 alias name.

CONSTRAINT constraint-name
Names the constraint for which you want to create a comment. If the constraint is not in the default schema, you must qualify the constraint name in the COMMENT ON statement with an alias name.

DATABASE
This statement writes the new comment to the database. If the ALIAS clause is omitted, the default database is used. The alias-name must be an alias specified by an ATTACH or CONNECT statement during this session.

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 alias name.

FUNCTION function-name
Names the function for which you want to create a comment. If the function is not in the default schema, you must qualify the function name in the COMMENT ON statement with an alias name.

INDEX index-name
PARTITION partition-name
Names the index and, optionally, a partition in the named 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 alias name.

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 (/).

MODULE module-name
Names the module for which you want to create a comment. If the module is not in the default schema, you must qualify the module name in the COMMENT ON statement with an alias name.

PROCEDURE procedure-name
Names the procedure for which you want to create a comment. If the procedure is not in the default schema, you must qualify the procedure name in the COMMENT ON statement with an alias name.

PROFILE profile-name
Names the profile for which you want to create a comment. If the profile is not in the default schema, you must qualify the profile name in the COMMENT ON statement with an alias name.

ROLE role-name
Names the role for which you want to create a comment. If the role is not in the default schema, you must qualify the role name in the COMMENT ON statement with an alias name.

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 alias name.

SEQUENCE sequence-name
Names the sequence for which you want to create a comment. If the sequence is not in the default schema, you must qualify the sequence name in the COMMENT ON statement with an alias name.

STORAGE MAP map-name
PARTITION partition-name
Names the storage map and, optionally, a vertical or horizontal partition within that storage map, for which you want to create a comment. If the storage map is not in the default schema, you must qualify the storage map name in the COMMENT ON statement with an alias name.

SYNONYM synonym-name
This performs the same function as the ALTER SYNONYM . . . COMMENT IS syntax. The synonym-name must be the name of an existing synonym. A database alias can be used to select a database other than the default database alias.

TABLE table-name
Names the table for which you want to create a comment. If the table is not in the default schema, you must qualify the table name in the COMMENT ON statement with an alias name.

table-name col-name
Names the table and the column or columns in that table for which you want to create a comment.

TRIGGER trigger-name
Names the trigger for which you want to create a comment. If the trigger is not in the default schema, you must qualify the trigger name in the COMMENT ON statement with an alias name.

USER user-name
Names the user (created with the CREATE USER statement) for which you want to create a comment. If the user is not in the default schema, you must qualify the user name in the COMMENT ON statement with an alias name.

VIEW view-name
Names the view for which you want to create a comment. If the view is not in the default schema, you must qualify the view name in the COMMENT ON statement with an alias name.

Usage Notes

- You must have ALTER privilege on the object, or in the case of constraints and triggers, you must have ALTER privilege on the parent object.
- 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. The total comment length must be less than 2 GB.
- The COMMENT ON TABLE statement can reference a table reserved in DATA DEFINITION mode.

Examples

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
```

Example 3: Adding a Comment to a Trigger

```
SQL> COMMENT ON TRIGGER EMPLOYEE_ID_CASCADE_DELETE IS
cont> 'When an employee is deleted from EMPLOYEES, delete'/
cont> 'corresponding records from the other tables in the'/
cont> 'database.';
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
                -- null for that department.
                  (UPDATE DEPARTMENTS D  SET D.MANAGER_ID = NULL
                   WHERE D.MANAGER_ID = EMPLOYEES.EMPLOYEE_ID)
                     FOR EACH ROW.
   .
   .
   .
Comment:       When an employees is deleted from EMPLOYEES, delete
               corresponding records from the other tables in the
               database.
```

Example 4: Adding Comments to Multiple Columns in a Table

```
SQL> COMMENT ON JOBS (JOB_CODE is 'Required column',
cont> WAGE_CLASS is 'Valid values are: 1, 2, 3, or 4');
SQL> SHOW TABLE (COLUMNS) JOBS;
Information for table JOBS
Columns for table JOBS:
Column Name                    Data Type        Domain
-----------                    ---------        ------
JOB_CODE                       CHAR(4)          JOB_CODE
 Comment:       Required column
 Missing Value: None
WAGE_CLASS                     CHAR(1)          WAGE_CLASS
 Comment:       Valid values are: 1, 2, 3, or 4
JOB_TITLE                      CHAR(20)         JOB_TITLE
 Missing Value: None
MINIMUM_SALARY                 INTEGER(2)       SALARY
MAXIMUM_SALARY                 INTEGER(2)       SALARY
```

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 (unless they are WITH HOLD 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-statement =

Arguments

AND CHAIN
When AND CHAIN is used, a new transaction is implicitly started using the same attributes as this committed transaction.

WORK
An optional keyword that has no effect on the COMMIT statement.

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, INSERT, UPDATE, REPLACE and TRUNCATE TABLE statements.
  - All changes made to the data definitions using the ALTER, CREATE, DROP, GRANT, REVOKE, RENAME, 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 use the COMMIT statement in an ATOMIC compound statement.
- The COMMIT statement may not be executed from a SQL function or trigger or any stored procedure called from a SQL function or trigger.

- The AND CHAIN clause is only permitted in a compound statement (that is, in a BEGIN . . . END block), or as the body of a stored procedure.
- When AND CHAIN is used a new transaction is implicitly started using the same attributes as the previously commited or rolled back transaction. Attributes such as READ WRITE, READ ONLY, RESERVING, EVALUATING, WAIT, and ISOLATION LEVEL are retained for the new transaction.

  Applications can use this new clause to simplify applications, since the complex transaction attributes need only be specified once.
- When the SET FLAGS option TRANSACTION_PARAMETERS is specified, a line of output is written to identify the chained transaction. Each SET TRANSACTION assigns a unique sequence number which is displayed after each transaction action line.

  ```
  ~T Restart_transaction (3) on db: 1, db count=1
  ```
- When the COMMIT statement is executed within a compound statement and no transaction is active, a success status (SQLSTATE or SQLCODE) is the result.

  However, if the COMMIT statement is executed in a single statement, it will result in an error. This behavior can be modified by setting the dialect to SQL92, SQL99, SQL2011, ORACLE LEVEL1, ORACLE LEVEL2 or ORACLE LEVEL3 or by using the SET QUIET COMMIT statement. Refer to the SET DIALECT and SET QUIET COMMIT statements for more details. For SQL Module Language or SQL pre-compiler applications, refer to the QUIET_COMMIT qualifier and the QUIET COMMIT clause in the module header.

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
```

Example 3: Using the AND CHAIN argument

The following simple example executes SET TRANSACTION once at the start of the procedure. Then periodically the transaction is committed and restarted using the COMMIT AND CHAIN syntax. This simplifies the application since there is only one definition of the transaction characteristics.

```
SQL> -- process table in batches
SQL>
SQL> set compound transactions 'internal';
SQL> set flags 'transaction,trace';
SQL>
SQL> begin
cont> declare :counter integer = 0;
cont> declare :emp integer;
cont>
cont> set transaction
cont>     read write
cont>     reserving employees for exclusive write;
cont>
cont> for :emp in 0 to 600
cont> do
cont>     begin
cont>     declare :id char(5)
cont>           default substring (cast (:emp+100000 as varchar(6))
cont>                              from 2 for 5);
cont>     if exists (select * from employees where employee_id = :id)
cont>     then
cont>         trace 'found: ', :id;
cont>         if :counter > 20
cont>         then
cont>             commit and chain;
cont>             set :counter = 1;
cont>         else
cont>             set :counter = :counter + 1;
cont>         end if;
cont>     end if;
cont>     end;
cont> end for;
cont>
cont> commit;
cont> end;
~T Compile transaction (1) on db: 1
~T Transaction Parameter Block: (len=2)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_WRITE (read write)
~T Start_transaction (1) on db: 1, db count=1
~T Rollback_transaction on db: 1
~T Compile transaction (3) on db: 1
~T Transaction Parameter Block: (len=14)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_WRITE (read write)
0002 (00002) TPB$K_LOCK_WRITE (reserving) "EMPLOYEES" TPB$K_EXCLUSIVE
~T Start_transaction (3) on db: 1, db count=1
~Xt: found: 00164
~Xt: found: 00165
.
.
.
```

COMMIT Statement

```
~Xt: found: 00185
~T Commit_transaction on db: 1
~T Prepare_transaction on db: 1
~T Restart_transaction (3) on db: 1, db count=1
~Xt: found: 00186
~Xt: found: 00187
.
.
.
~Xt: found: 00435
~Xt: found: 00471
~T Commit_transaction on db: 1
~T Prepare_transaction on db: 1
SQL>
```

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 Rdb Guide to SQL Programming* for a conceptual description of compound statements and their relationship to multistatement procedures.

Environment

You can use a compound statement:

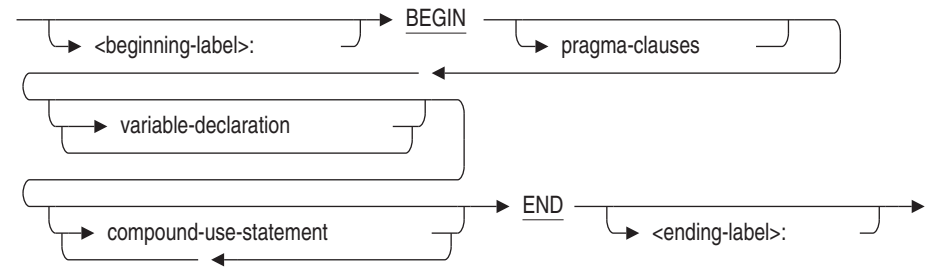
- In interactive SQL, as a way to test syntax and prototype compound statements for use with programs.
- In embedded SQL, as part of a host language program to be processed with the SQL precompiler.

Compound Statement

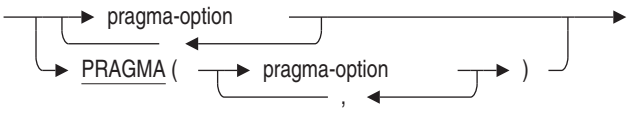
- In SQL module language, as part of a multistatement procedure in an SQL module file to be processed with the SQL module processor.
- In dynamic SQL, to prepare and execute compound statements.

Format

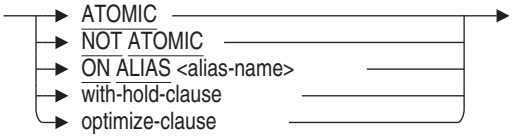
compound-statement =



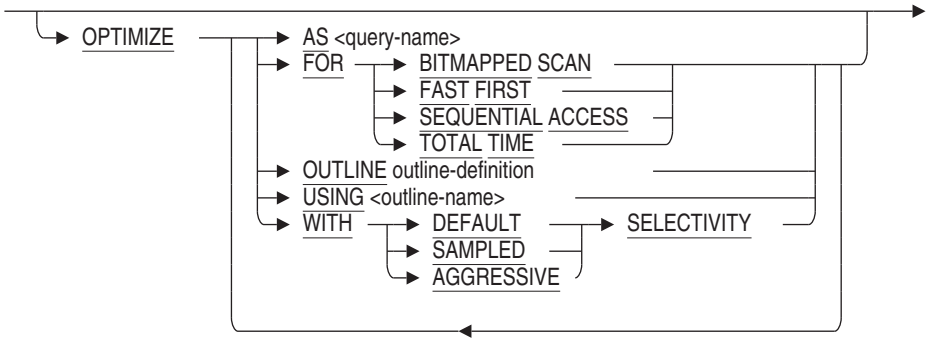
pragma-clauses =



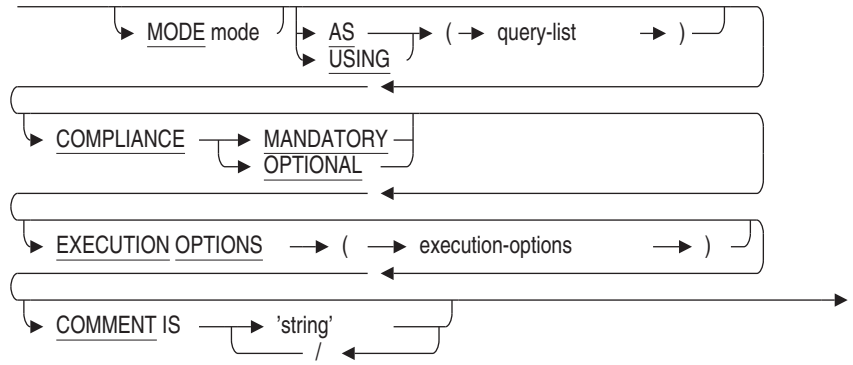
pragma-option =



optimize-clause =



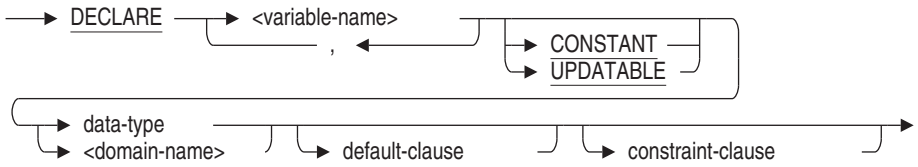
outline-definition =



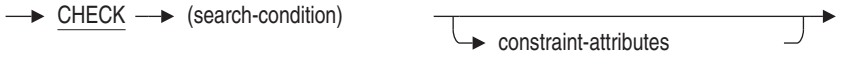
with-hold-clause =

Compound Statement

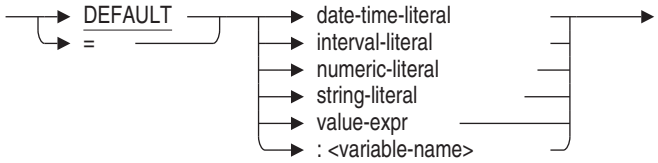
variable-declaration =



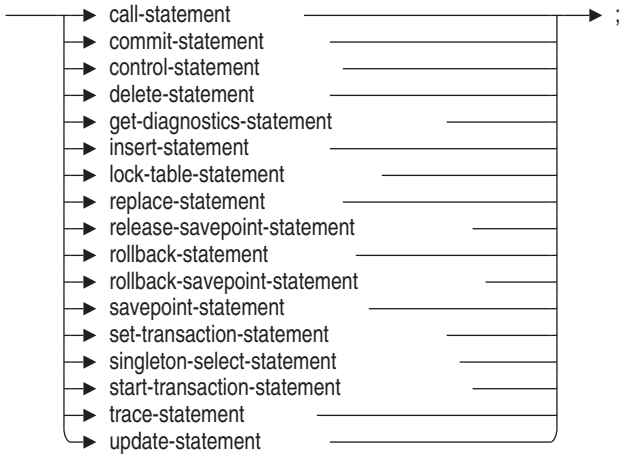
constraint-clause =

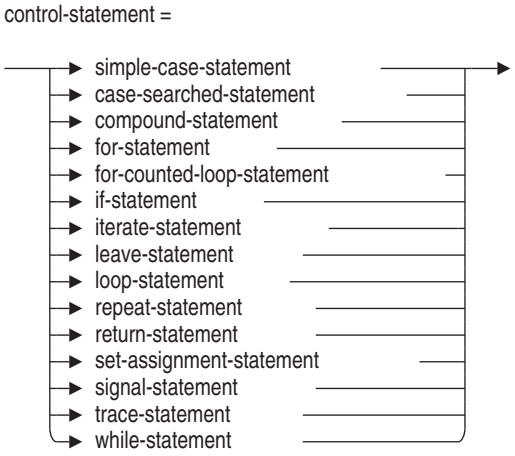


default-clause =



compound-use-statement =





Arguments

ATOMIC
NOT ATOMIC

Controls whether or not SQL statements in the 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, START 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 property of an ATOMIC block is that all statements succeed, or all statements fail, and this can not be guaranteed if a transaction is started or ended during the block.

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;).

beginning-label:
Assigns a name to a block. You use the label with the LEAVE or ITERATE statements 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.

call-statement
Invokes an external or stored procedure. See the CALL Statement for Compound Statements for a complete description.

case-searched-statement
See the CASE (Searched) Control Statement 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. The AND CHAIN clause can also be used to start a new transaction.

See the COMMIT Statement for a complete description.

compound-statement
Lets you nest compound statements in another compound statement.

compound-use-statement
Identifies the SQL statements allowed in a compound statement block.

CONSTANT
CONSTANT changes the variable into a declared constant that can not 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 can not be used as the target of an assignment or be passed as an expression to a procedure's INOUT or OUT parameter.

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.

default-clause
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.

delete-statement
Deletes a row from a table or view.

See the DELETE 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.

for-counted-loop-statement
See the FOR (Counted) Control Statement for a complete description.

for-statement
See the FOR Control Statement for a complete description.

Compound Statement

get-diagnostics-statement
Retrieves diagnostic information for the previously executed statement.

See the GET DIAGNOSTICS Statement for a complete description.

if-statement
See the IF Control 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.

leave-statement
See the LEAVE Control Statement for a complete description.

lock-table-statement
See the LOCK TABLE Statement for a complete description.

loop-statement
See the LOOP Control Statement for a complete description.

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
cont> SELECT EMPLOYEE_ID INTO :x FROM db1.EMPLOYEES
cont> WHERE EMPLOYEE_ID='00164';
cont> END;
SQL> PRINT :x;
 X
 00164
```

OPTIMIZE AS request
Assigns a name to the compound statement.

OPTIMIZE OUTLINE outline-definition
The OPTIMIZE OUTLINE clause declares a temporary query outline to be used with the select expression.

See the CREATE OUTLINE Statement for more information on defining an outline.

OPTIMIZE USING outline-name
Names the query outline to be used with the compound statement, even if the outline ID for the query and for the outline are different.

OPTIMIZE WITH
Selects one of three optimization controls: DEFAULT (as used by previous versions of Oracle Rdb), AGGRESSIVE (assumes smaller numbers of rows will be selected), and SAMPLED (which uses literals in the query to perform preliminary estimation on indices).

PRAGMA pragma-options
These options may only be specified on the outermost BEGIN statement. The exception is ATOMIC and NOT ATOMIC.

repeat-statement
See the REPEAT Control Statement for a complete description.

release-savepoint-statement
The RELEASE SAVEPOINT Statement destroys the named savepoint established by the SAVEPOINT statement.

See the RELEASE SAVEPOINT Statement for a complete description.

replace-statement
Adds (or replaces) a new row, or a number of rows, to a table or view. For compound statements, SQL restricts the REPLACE statement to database insert operations in a single database.

See the REPLACE Statement for a complete description.

return-statement
Returns the result for stored functions. See the RETURN Control 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. The AND CHAIN clause can also be used to start a new transaction.

See the ROLLBACK Statement for a complete description.

rollback-savepoint-statement
The ROLLBACK TO SAVEPOINT Statement destroys the named savepoint established by the SAVEPOINT statement and removes all database changes made from the time the SAVEPOINT statement established the named savepoint.

See the ROLLBACK TO SAVEPOINT Statement for a complete description.

savepoint-statement
The SAVEPOINT Statement establishes a marker in the current transaction that allows the programmer to undo part of the transaction (using ROLLBACK TO SAVEPOINT) without resorting to a full transaction ROLLBACK.

See the SAVEPOINT Statement for a complete description.

set-assignment-statement
See the SET Control Statement for a complete description.

set-transaction-statement
Starts a transaction and specifies its characteristics.

See the SET TRANSACTION Statement for a complete description.

signal-statement
See the SIGNAL Control Statement for a complete description.

simple-case-statement
See the CASE (Simple) Control Statement for a complete description.

singleton-select-statement
Specifies a one-row result table.

See the SELECT Statement: Singleton Select for a complete description.

start-transaction-statement
See the START TRANSACTION Statement for a complete description.

trace-statement
Writes values to the trace log file. See the TRACE Control Statement for a complete description.

UPDATABLE
UPDATABLE is the default (versus CONSTANT) 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), or as a procedure's OUT or INOUT parameter.

update-statement
Modifies a row in a table or view.

See the UPDATE Statement for a complete description.

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.

while-statement
See the WHILE Control Statement for a complete description.

WITH HOLD
Can be applied to a table cursor so that it remains open across COMMIT and ROLLBACK actions. See the Usage Notes for more information.

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.
- 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 of a domain used for the variable.
- 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 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.
- If an outline exists, Oracle Rdb will use the outline specified in the OPTIMIZE USING clause unless one or more of the directives in the outline cannot be followed. SQL issues an error message if the 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.
- Variables declared within a compound statement (BEGIN . . . END) can include a CHECK constraint to prevent out of range assignments to variables.
- The constraint-clause is applied to all variables listed in DECLARE. The keyword VALUE can be used as a placeholder for the variable name with SQL correctly applying the constraint to all variables.
- Only the NOT DEFERRABLE and INITIALLY IMMEDIATE syntax is supported for variable constraints. This is also the default if no constraint-attributes are specified.
- A runtime error is signalled if the constraint is violated. This error will include the name of the variable.

- When a DEFAULT is not used in the declare statement the contents of the variable are undefined. Therefore, any variable that uses a CHECK constraint must also provide a DEFAULT clause to ensure that the variable's value is consistent.
- When the WITH HOLD clause is applied to the outer BEGIN of a compound statement it forces WITH HOLD cursor semantics for all FOR cursor loops within this compound statement.
- The WITH HOLD semantic allows, for instance, applications that processes large numbers of rows to commit frequently to reduce lock contention and size of recovery journals.

Note

These semantics are not inherited by procedures called by the CALL statement, nor by SQL functions executed within statements in the compound statement.

- The clause WITH HOLD PRESERVE NONE is a way to specify the default action, which is to always close cursors upon commit.
- The default action, or when WITH HOLD PRESERVE NONE is specified, is to forbid the use of the COMMIT, ROLLBACK, SET TRANSACTION and START TRANSACTION statements within a FOR cursor loop.

  If WITH HOLD PRESERVE ON COMMIT, WITH HOLD PRESERVE ON ROLLBACK or WITH HOLD PRESERVE ALL are used then these statements will be allowed within a FOR cursor loop.

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
```

Example 4: Using the CHECK constraint

This example shows the use of a CHECK constraint to prevent illegal values being assigned to control variables for a REPEAT loop. The singleton SELECT will actually return zero to the local variable P which will cause a variable validation to fail.

```
SQL> begin
cont> declare :v integer = 0 check (value is not null);
cont> declare :p integer = 1 check (value is not null and value <> 0);
cont>
cont> repeat
cont>     select count(*) into :p
cont>     from employees
cont>     where employee_id = '00000';
cont>     set :v = :v + :p;
cont> until :v > 1000
cont> end repeat;
cont> end;
%RDB-E-NOT_VALID, validation on field P caused operation to fail
```

Example 5: Using the WITH HOLD clause

The following example shows the use of the WITH HOLD PRESERVE ON COMMIT clause in a procedure which purges old data from the AUDIT_ HISTORY table. It commits the transaction every 100 rows (:MAX_UNIT).

```
SQL> declare transaction read only;
SQL> set flags 'TRACE';
SQL> set compound transactions 'internal';
SQL> declare :purge_date date;
SQL> accept :purge_date prompt 'Purge date for AUDIT_HISTORY? ';
Purge date for AUDIT_HISTORY? 1-jan-1989
SQL>
SQL> begin
cont>     pragma (with hold preserve on commit)
cont> declare :max_unit constant integer = 100;
cont> declare :rc integer = :max_unit;
cont>
cont> set transaction read write;
cont>
cont> for :ah as table cursor ah_cursor
cont>     for select *
cont>         from audit_history
cont>         where job_start < :purge_date
cont>     do
cont>         delete from audit_history
cont>             where current of ah_cursor;
cont>         if :rc = 0
cont>         then
cont>             commit and chain;
cont>             set :rc = :max_unit;
cont>         else
cont>             set :rc = :rc - 1;
cont>         end if;
cont> end for;
cont> get diagnostics :rc = ROW_COUNT;
cont> commit;
cont> trace 'Processed rows: ', :rc;
cont> end;
~Xt: Processed rows: 2192
```

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.

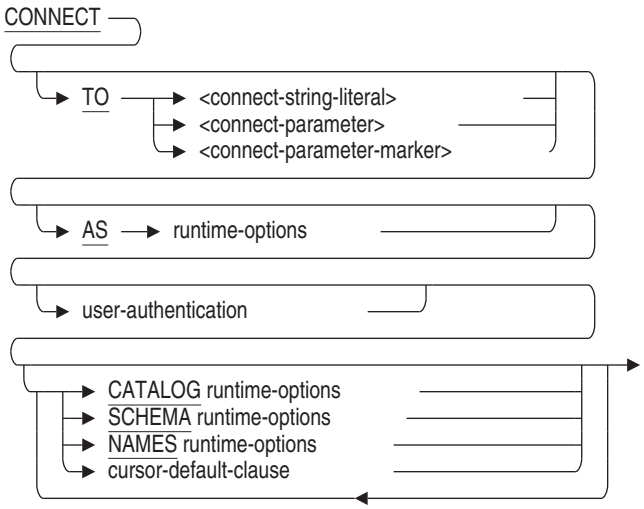
The DISCONNECT statement detaches from databases, ends the transactions in the connections that you specify, and rolls back all the changes you made since those transactions began.

Environment

You can use the CONNECT statement:

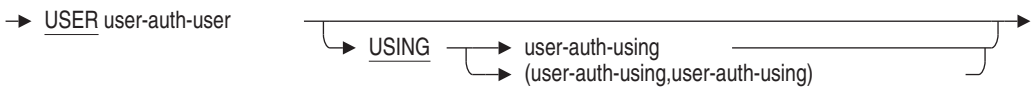
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

Format

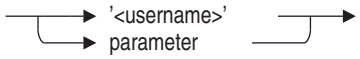


CONNECT Statement

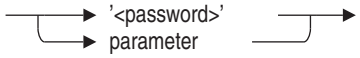
user-authentication =



user-auth-user =



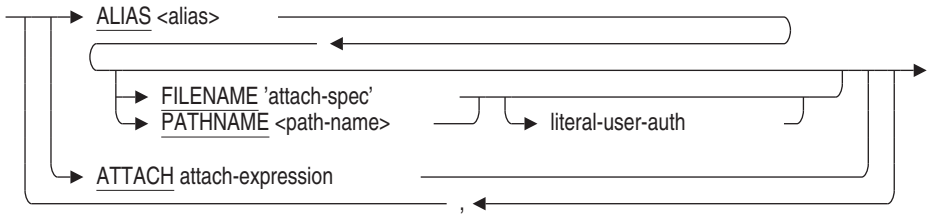
user-auth-using =



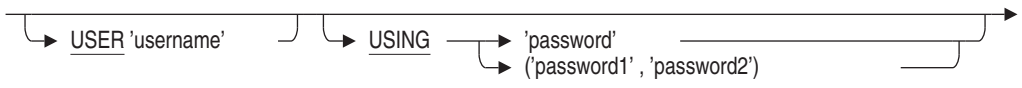
connect-string-literal =

connect-expression =

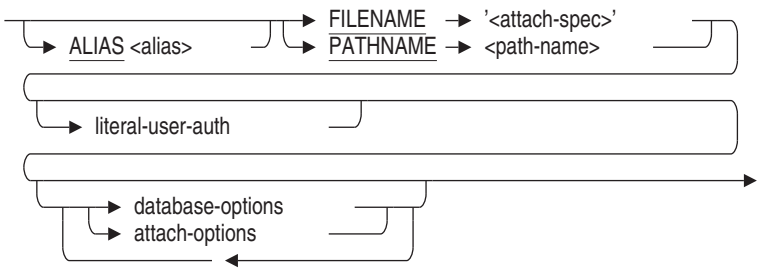
db-specification =



literal-user-auth =

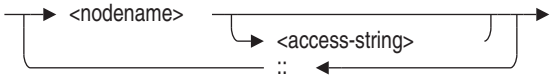


attach-expression =

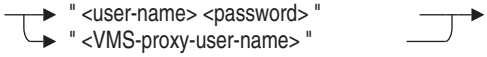


attach-spec =

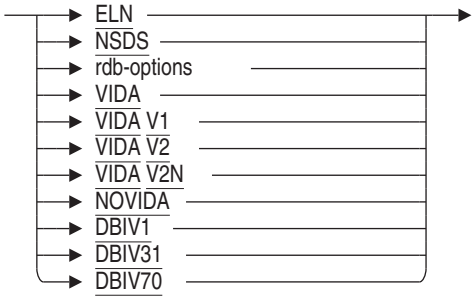
node-spec =



access-string =

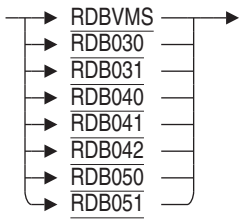


database-options =

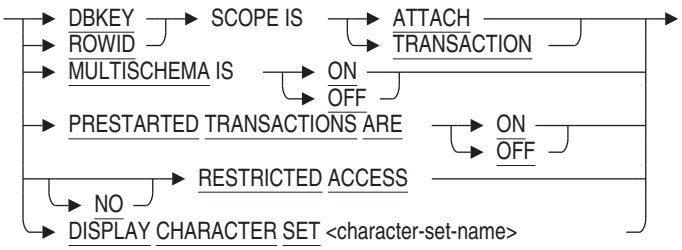


CONNECT Statement

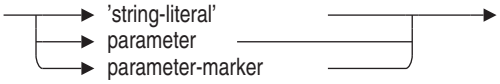
rdb-options =



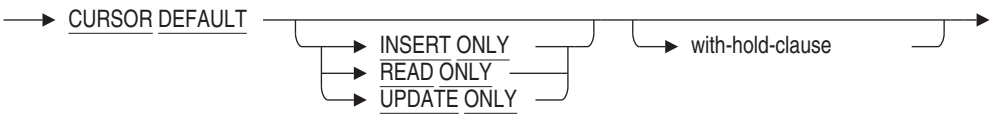
attach-options =



runtime-options



cursor-default-clause =



with-hold-clause =

Arguments

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.

AS runtime-options

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
```

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.

CATALOG runtime-options

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.

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.

FILENAME 'attach-spec'
A quoted string containing full or partial information needed to access a database.

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 2.2.8.1.

'string-literal'
parameter-marker
host-variable
Specifies a value, possibly specified at runtime, that is used by various CONNECT clauses.

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.

NAMES runtime-options
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 Section 2.1 .

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.

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.

SCHEMA runtime-options
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.

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.

USER 'username'
USER parameter
A character string literal or parameter that specifies the operating system user name that the database system uses for privilege checking. The parameter can be either a CHAR or VARCHAR value.

USING 'password'
USING parameter
USING ('password','password')
USING (parameter,parameter)
A character string literal or parameter that specifies the user's password for the user name specified in the USER clause. If more than one password is used, either or both can be parameters. The parameters can be either a CHAR or VARCHAR variables.

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.

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.
- 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*.
- For DISPLAY CHARACTER SET, the specified character set must contain ASCII characters. See Section 2.1.5 for a list of allowable character sets. The option can be a literal, a parameter, or a parameter marker.
- The character set also specifies the character set for the SQLNAME field in SQLDA and SQLDA2 for statements without an explicit database context.
- 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;
        RDB$DEFAULT_CONNECTION
->      TEST
SQL> show connections TEST;
Connection: TEST
Default alias is RDB$DBHANDLE
Default catalog name is RDB$CATALOG
Default schema name is SMITH
Dialect: SQLV40
Default character unit: OCTETS
Keyword Rules: SQLV40
View Rules: SQLV40
Default DATE type: DATE VMS
Quoting Rules: SQLV40
Optimization Level: DEFAULT
Hold Cursors default: WITH HOLD PRESERVE NONE
Quiet commit mode: OFF
Compound transactions mode: EXTERNAL
Default character set is DEC_MCS
National character set is DEC_MCS
Identifier character set is DEC_MCS
Literal character set is DEC_MCS
Display character set is UNSPECIFIED

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 database 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 Statements

Creates the database object.

Usage Notes

The following notes apply to all CREATE statements except CREATE DATABASE.

- You cannot execute the CREATE statement when any of the LIST, DEFAULT or RDB$SYSTEM storage areas are set to read-only. You must first set these storage areas to read/write. Note that in some databases RDB$SYSTEM will also be the default and list storage area.
- You must execute the CREATE 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 CREATE 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
```

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 using the ADD CACHE clause of the ALTER DATABASE statement.

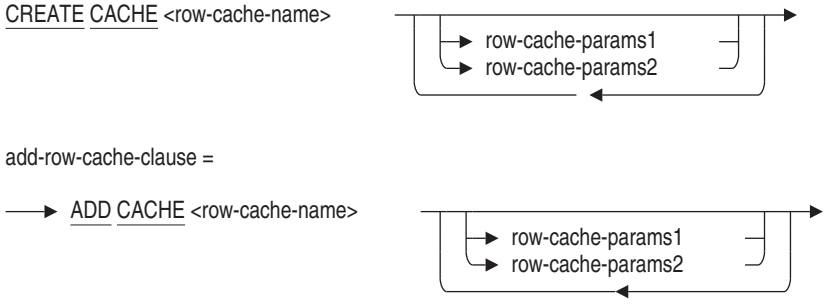
Creates a row cache that allows frequently referenced rows to remain in memory even when the associated page has been transferred back to disk. This saves in memory usage because only the more recently referenced rows are cached versus caching the entire buffer.

See the ALTER DATABASE Statement and the CREATE DATABASE Statement for more information regarding the row cache areas.

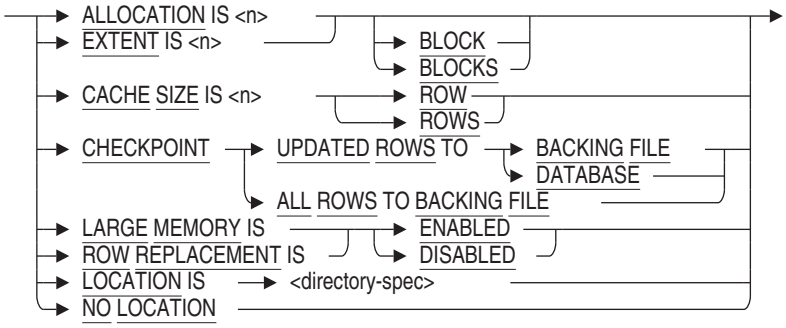
Environment

You can use the CREATE CACHE clause only within a CREATE DATABASE or IMPORT statement. You can use the ADD CACHE clause only within the ALTER DATABASE statement.

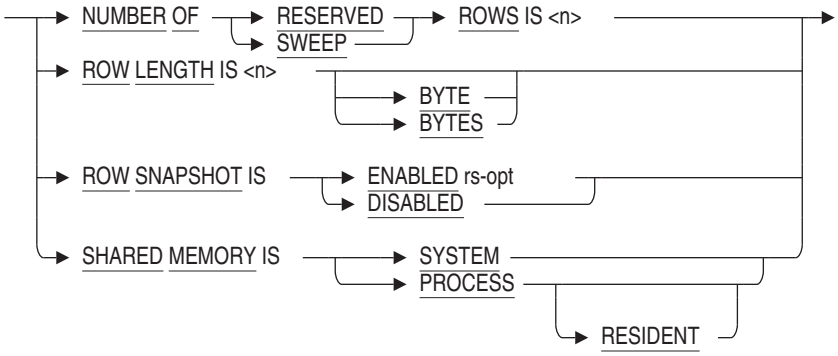
Format



row-cache-params1 =



row-cache-params2 =



rs-opt =

Arguments

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.

This clause is ignored if the row cache is defined to checkpoint to the database.

CREATE CACHE Clause

CACHE row-cache-name
Creates a row cache.

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

CHECKPOINT ALL ROWS TO BACKING FILE
CHECKPOINT UPDATED ROWS TO BACKING FILE
CHECKPOINT UPDATED ROWS TO DATABASE
Specifies the source records and target for checkpoint operations for the row cache. If ALL ROWS is specified, then the records written during each checkpoint operation are both the modified and the unmodified rows in the row cache. If UPDATED ROWS is specified, then just the modified rows in the row cache are checkpointed each time.

If the target of the checkpoint operation is BACKING FILE, then the row cache server (RCS) process writes the row cache entries to the backing (.rdc) files. The row cache LOCATION, ALLOCATION, and EXTENT clauses are used to create the backing files. Upon recovery from a node failure, the database recovery process is able to repopulate the row caches in memory from the rows found in the backing files.

If the target is DATABASE, then the updated rows (only UPDATED ROWS is allowed) are written back to the database. The row cache LOCATION, ALLOCATION, and EXTENT clauses are ignored. Upon recovery from a node failure, the database recovery process has no data on the contents of the row cache. Therefore, it does not repopulate the row caches in memory.

This CHECKPOINT clause overrides the database-level CHECKPOINT clause.

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.

This clause is ignored if the row cache is defined to checkpoint to the database.

LARGE MEMORY IS ENABLED
LARGE MEMORY IS DISABLED
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.

Use the LARGE MEMORY IS ENABLED clause 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 does not fit in the virtual address space.

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

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 at checkpoint time. Specify a device name and directory name only, enclosed within single quotation marks. 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.

This clause is ignored if the row cache is defined to checkpoint to the database.

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

This clause is ignored if the row cache is defined to checkpoint to the database.

NUMBER OF RESERVED ROWS IS n
Specifies the maximum number of cache rows that each user can reserve for insertion into the cache. Processes reserve, or allocate, entries in a cache to be used when inserting rows into the cache. To improve efficiency, multiple entries are reserved at one once. Once a user's reserved list becomes depleted, Oracle Rdb attempts to reserve another group of entries. The default is 20 rows.

This value is also used when searching for available slots in a row cache. The entire row cache is not searched on the initial pass. This value specifies the maximum number of rows that are searched for a free slot. If at least one free slot is found, the insert operation can proceed. If no free slots are found in this initial search, Oracle Rdb continues searching through the cache until it finds a free slot.

NUMBER OF SWEEP ROWS IS n
Specifies the number of modified rows that will be written from the row cache back to the database by the **row cache server (RCS)** process during a sweep operation. When the RCS is notified that a cache is "full" of modified data, the RCS starts a sweep to make space available in the cache for subsequent transactions to be able to insert rows into the cache. Oracle Corporation recommends that you initially specify the number of sweep rows to be approximately 5 percent of the total number of rows in the cache. Then monitor performance and adjust the number of sweep rows, if necessary. Allowable values must be in the range 2 through 524288. If not specified, the default is 3,000 rows.

ROW LENGTH IS n BYTES
Specifies the length of each row stored in the row cache. Rows (or index nodes) are not cached if they are too large for the row cache entry or the rows are fragmented on a page. The ROW LENGTH is implicitly rounded up to a multiple of 4 bytes, allowing the entry to be longword aligned.

The maximum row length in a row cache is 65535 bytes which is the largest row supported by Oracle Rdb. This length for a table row includes all columns, null bit vector, Rdb header, and padding; similarly an index node includes the node, Rdb header and padding.

When the name of the cache matches the name of an existing logical area, ADD CACHE will calculate ROW LENGTH from the size of the table row, or the size of the index node (for SORTED RANKED, or UNIQUE SORTED indices). This cache is known as a logical area cache.

If the ROW LENGTH clause is omitted, Oracle Rdb will attempt to estimate a ROW LENGTH using the length of the table row or the size of the index node (SORTED RANKED, or SORTED UNIQUE only). Otherwise, it will default to 256 bytes.

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 becomes full. When ROW REPLACEMENT IS DISABLED, rows are not

replaced when the row cache is full. The type of row replacement policy depends upon the application requirements for each cache.

The default is ENABLED.

ROW SNAPSHOT IS DISABLED
Disables storing snapshot copies of rows within the cache.

ROW SNAPSHOT IS ENABLED (CACHE SIZE IS n ROWS)
The ROW SNAPSHOT IS ENABLED (CACHE SIZE IS n ROWS) option enables storage of snapshot copies of rows within the cache and specifies the number of snapshot "slots" to allocate for the cache.

The default for new caches, and existing caches is to have this feature disabled.

If you do not specify the CACHE SIZE clause for the ROW SNAPSHOT IS ENABLED option, Oracle Rdb creates a cache that can contain up to 1000 snapshot rows.

SHARED MEMORY IS SYSTEM
SHARED MEMORY IS PROCESS
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 operating system maps a row cache 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.

SHARED MEMORY IS PROCESS RESIDENT
The SHARED MEMORY clause determines whether database root global sections (including global buffers when enabled) or whether the cache global sections are created in system space or process space. The RESIDENT option extends the PROCESS option by making the global section memory resident.

Usage Notes

- If the name of the row cache is the same as any logical area (for example a table name, index 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.
- Neither ADD nor CREATE CACHE clause assigns the row cache 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 (some additional overhead and rounding up to page boundaries is performed by the database system).
- The row cache is shared by all processes attached to the database on any one 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 nodes must equal 1 or the system is running an OpenVMS Galaxy configuration and NUMBER OF CLUSTER NODE . . . (SINGLE INSTANCE) is enabled.
- Use the SHOW CACHE statement to view information about a cache.
- Each row cache defined in a database can be designated to allow a specified number of snapshot rows to be stored in the cache. The number of snapshot rows allowed is specified in addition to the number of live database rows that can be stored in the cache. Because many versions of a row may be stored in the snapshot portion of the cache, the number of snapshot rows and live cache rows are specified independently. As the snapshot portion of the row cache is effectively an extension of the row cache itself, other attributes of the cache are shared with the snapshot portion.

- The snapshot portion of a row cache may be larger (may contain more rows) or smaller (may contain fewer rows) than the live portion of the row cache. Because application and workload behavior determines the number of database rows that are modified and the relative transaction length, it is not reasonable to make specific recommendations for sizing the snapshot portions caches for all application and database types. However, it is expected that the ratio of the size of the snapshot cache to the main cache may be similar to the ratio of the database snapshot storage area to the live storage area. If an application has long running transactions and active read-write transactions modifying cached data, many snapshot copies of the modified data may need to be maintained. This can require caches with many snapshot rows for those caches with heavy update activity.
- Very large memory (VLM) allows Oracle Rdb to use as much physical memory as is available on your system and to dynamically map it to the virtual address space of database users.

  The LARGE MEMORY parameter of the ALTER DATABASE...ADD CACHE statement is used to specify that row caches are to be created in very large memory.

  It is important that you consider the amount of memory available on your system before you start using VLM for row cache. You can use the DCL command SHOW MEMORY/PHYSICAL to check the availability and usage of physical memory. This command displays information on how much memory is used and how much is free.
- To enable or disable SHARED MEMORY IS PROCESS RESIDENT or LARGE MEMORY, the process executing the command must be granted the VMS$MEM_RESIDENT_USER rights identifier. When this feature is enabled, then the process that opens the database must also be granted the VMS$MEM_RESIDENT_USER identifier. Oracle recommends that the RMU/OPEN command be used when utilizing this feature.

Examples

Example 1: Creating a row cache

This example creates a database, creates a row cache, and assigns the row cache 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
        Working Set Count:     10
        Reserved Rows:         20
        Allocation:            100 blocks
        Extent:                100 blocks
SQL> SHOW STORAGE AREA area1

     AREA1
         Access is:      Read write
         Page Format:    Uniform
         Page Size:      4 blocks
         Area File:      USER2:[ADMIN.DATABASE]AREA1.RDA;1
         Area Allocation:          700 pages
         Extent:         Enabled
         Area Extent Minimum:      96 pages
         Area Extent Maximum:      9996 pages
         Area Extent Percent:      20 percent
         Snapshot File:  USER2:[ADMIN.DATABASE]AREA1.SNP;1
         Snapshot Allocation:      100 pages
         Snapshot Extent Minimum:  96 pages
         Snapshot Extent Maximum:  9996 pages
         Snapshot Extent Percent:  20 percent
         Locking is Row Level
         Using Cache TEST1
No database objects use Storage Area AREA1
SQL> DISCONNECT ALL;
```

Example 2: Creating and modifying various caches

The cache named CUSTOMER_STATUS is created with 88000 cache entries for storage of live database rows and 7000 rows for storage of snapshot copies. This cache is also configured to be memory resident. Note that no row length is specified; this allows ADD CACHE to fetch the actual row length from the table named CUSTOMER_STATUS.

The cache named MACHINE_FLOW_IDX_1 is created with 5000 cache rows for storage of live database rows and 12000 rows for storage of snapshot copies. This cache is set to disallow replacement of rows in the cache. Note that no row length is specified; this allows ADD CACHE to fetch the actual index node size from the index named MACHINE_FLOW_IDX_1.

The cache named SALES_CALLS is created with 3000 cache rows for storage of live database rows and, using the default because an explicit count was not specified, 1000 rows for storage of snapshot copies.

The cache named CUSTOMER_ORDER does not specify ROW SNAPSHOT IS ENABLED so no snapshot row copies will be stored in this cache. Any checkpoint will write back the updated rows to the database.

The cache named SALES is modified to disable storage of snapshot rows in cache.

The cache named CLEARING is modified to enable storage of snapshot rows in the cache with a snapshot cache size of 12300 rows.

```
SQL> ALTER DATABASE FILENAME HDB_DB
cont>     ADD CACHE CUSTOMER_STATUS
cont>         CACHE SIZE IS 88000 ROWS
cont>         ROW SNAPSHOT IS ENABLED (CACHE SIZE IS 7000 ROWS)
cont>         SHARED MEMORY IS PROCESS RESIDENT
cont>     ADD CACHE MACHINE_FLOW_IDX_1
cont>         CACHE SIZE IS 5000 ROWS
cont>         ROW REPLACEMENT IS DISABLED
cont>         ROW SNAPSHOT IS ENABLED (CACHE SIZE IS 12000 ROWS)
cont>     ADD CACHE SALES_CALLS
cont>         CACHE SIZE IS 3000 ROWS
cont>         ROW SNAPSHOT IS ENABLED
cont>     ADD CACHE CUSTOMER_ORDER
cont>         CACHE SIZE IS 9000 ROWS
cont>         CHECKPOINT UPDATED ROWS TO DATABASE
cont>     ALTER CACHE SALES
cont>         ROW SNAPSHOT IS DISABLED
cont>     ALTER CACHE CLEARING
cont>         ROW SNAPSHOT IS ENABLED (CACHE SIZE IS 12300 ROWS)
cont> ;
SQL>
```

The following SHOW CACHE statements will show all the specified and defaulted attributes for caches added during the ALTER DATABASE statement.

```
SQL> ATTACH 'FILENAME HDB_DB';
SQL>
SQL> SHOW CACHE CUSTOMER_STATUS

     CUSTOMER_STATUS
        Cache Size:            88000 rows
        Row Length:            528 bytes
        Row Replacement:       Enabled
        Shared Memory:         Process Resident
        Large Memory:          Disabled
        Window Count:          100
        Working Set Count:     10
        Reserved Rows:         20
        Allocation:            100 blocks
        Extent:                100 blocks
        Snapshot in Cache:     Enabled
        Snapshot Cache Size:   7000 rows
SQL> SHOW CACHE MACHINE_FLOW_IDX_1

     MACHINE_FLOW_IDX_1
        Cache Size:            5000 rows
        Row Length:            3000 bytes
        Row Replacement:       Disabled
        Shared Memory:         Process
        Large Memory:          Disabled
        Window Count:          100
        Working Set Count:     10
        Reserved Rows:         20
        Allocation:            100 blocks
        Extent:                100 blocks
        Snapshot in Cache:     Enabled
        Snapshot Cache Size:   12000 rows
SQL> SHOW CACHE SALES_CALLS

     SALES_CALLS
        Cache Size:            3000 rows
        Row Length:            128 bytes
        Row Replacement:       Enabled
        Shared Memory:         Process
        Large Memory:          Disabled
        Window Count:          100
        Working Set Count:     10
        Reserved Rows:         20
        Allocation:            100 blocks
        Extent:                100 blocks
        Snapshot in Cache:     Enabled
        Snapshot Cache Size:   1000 rows
SQL> SHOW CACHE CUSTOMER_ORDER
```

```
CUSTOMER_ORDER
   Cache Size:            9000 rows
   Row Length:            148 bytes
   Row Replacement:       Enabled
   Shared Memory:         Process
   Large Memory:          Disabled
   Window Count:          100
   Working Set Count:     10
   Reserved Rows:         20
   Allocation:            100 blocks
   Extent:                100 blocks
       - Checkpoint updated rows to database
```

CREATE CATALOG Statement

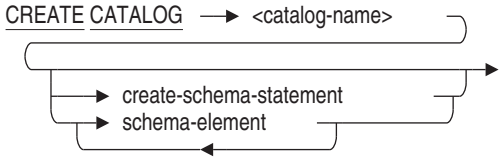
Creates a name for a group of schemas in a multischema database.

Environment

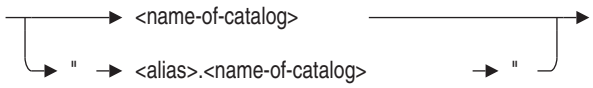
You can use the CREATE CATALOG statement:

- 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

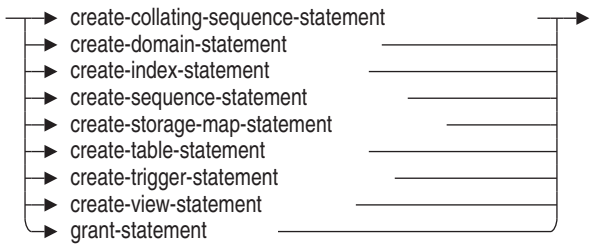
Format



catalog-name =



schema-element =



Arguments

"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.

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 2.2.3.

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 'SQL99' 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 'SQL99';
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

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:

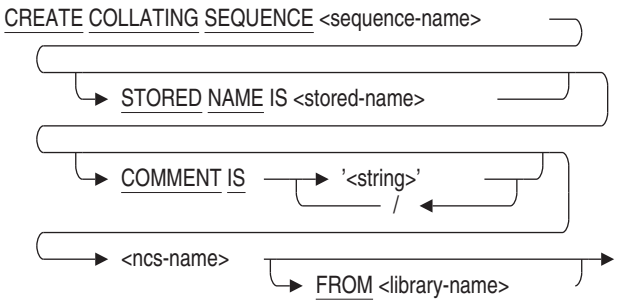
- CREATE DOMAIN . . . COLLATING SEQUENCE
- DROP COLLATING SEQUENCE
- ALTER DOMAIN . . . COLLATING SEQUENCE
- SHOW COLLATING SEQUENCE

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



Arguments

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 (/).

FROM library-name
Specifies the name of an NCS library other than the default. The default NCS library is SYS$LIBRARY:NCS$LIBRARY.

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.

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.

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.
  - 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 DATABASE . . . 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.
- 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)
- 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.
- Use COMMENT ON COLLATING SEQUENCE to change the comment for the collating sequence.

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.

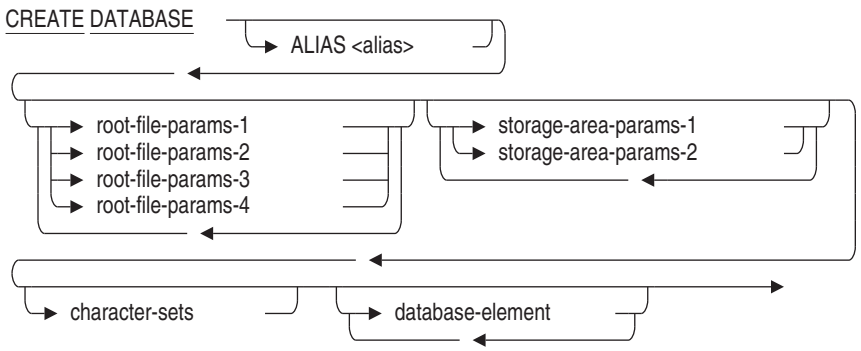
  Unlike the same statements outside a CREATE DATABASE statement, database elements do not use statement terminators. The first statement terminator that SQL encounters ends the CREATE DATABASE statement. Later CREATE or GRANT statements are not within the scope of the CREATE DATABASE statement.
- The database default character set and national character set. For information regarding identifier character sets, database default character sets, and national character sets, see Section 2.1.5, Section 2.1.3, and Section 2.1.7, respectively.

Environment

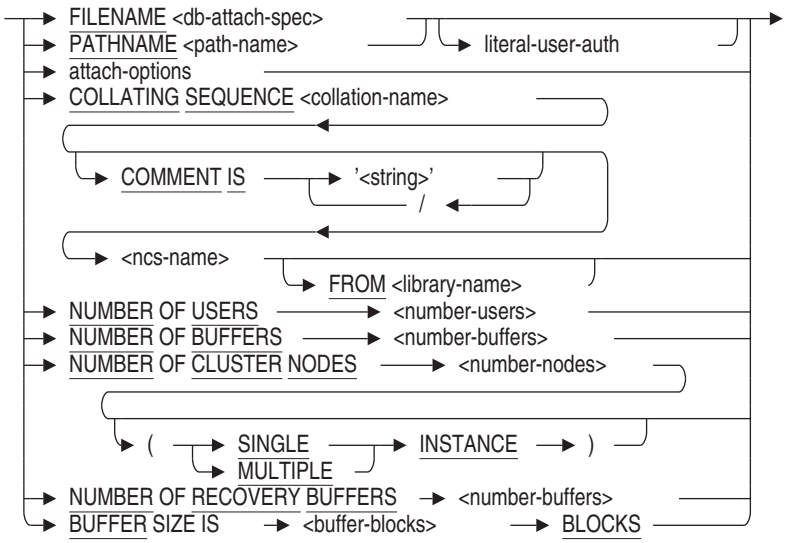
You can use the CREATE DATABASE statement:

- 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

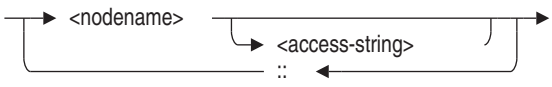


root-file-params-1 =

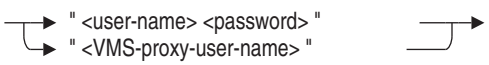


db-attach-spec =

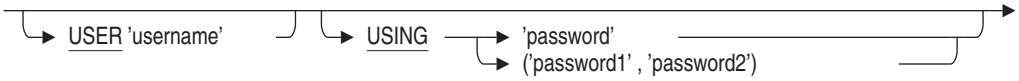
node-spec =



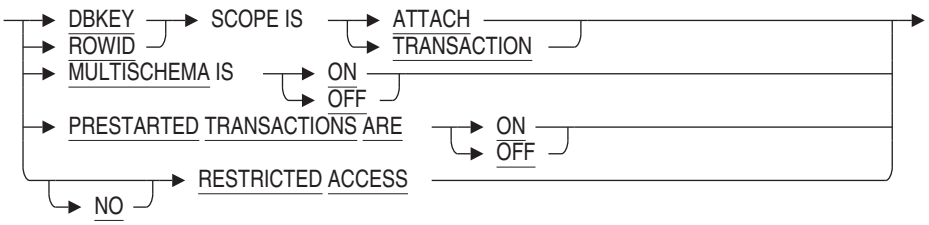
access-string =



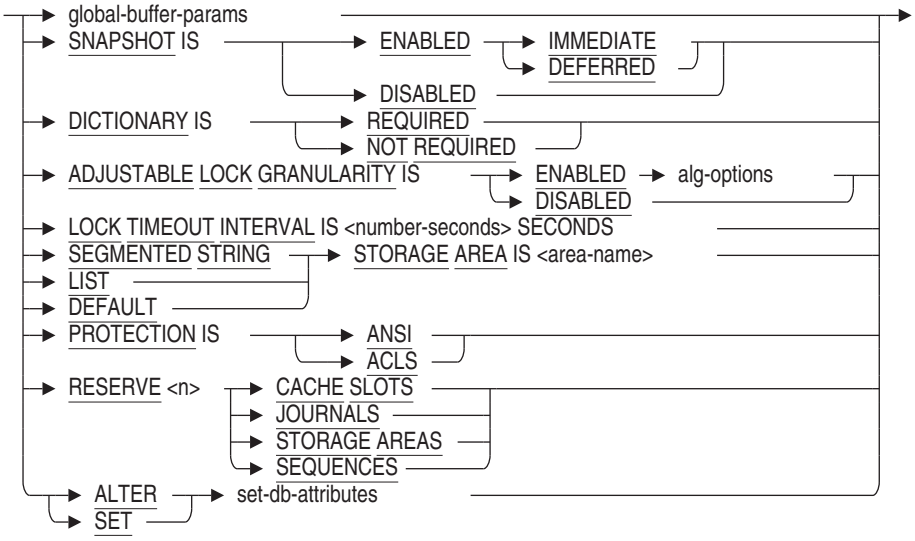
literal-user-auth =



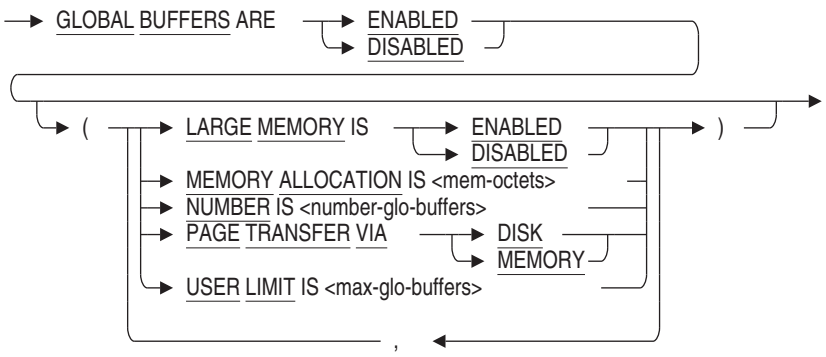
attach-options =



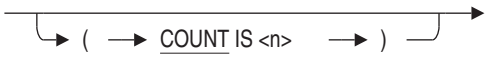
root-file-params-2 =



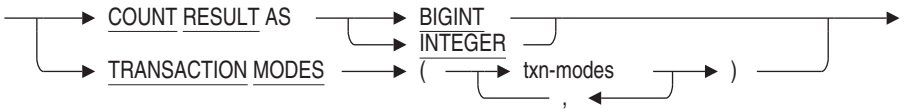
global-buffer-params=



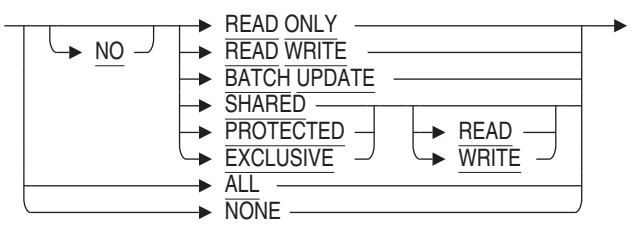
alg-options =



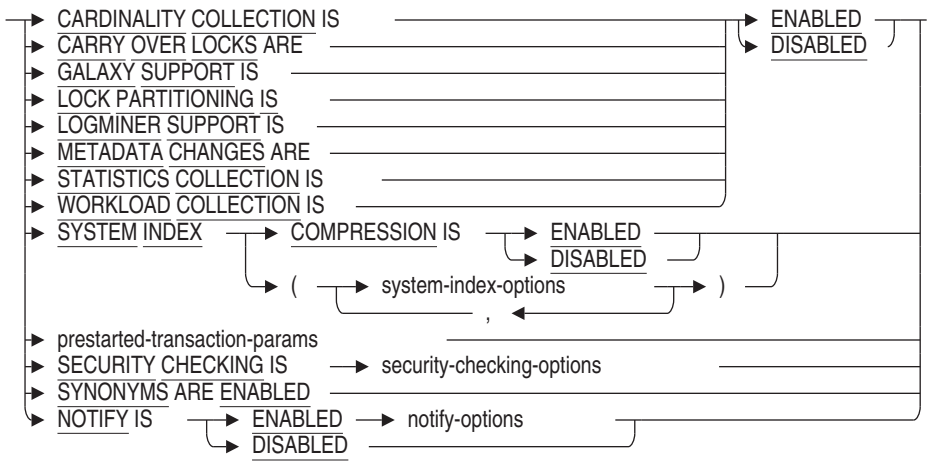
set-db-attributes =



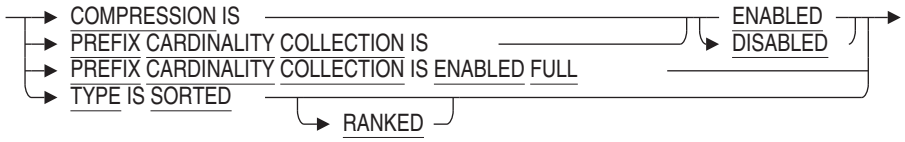
txn-modes =



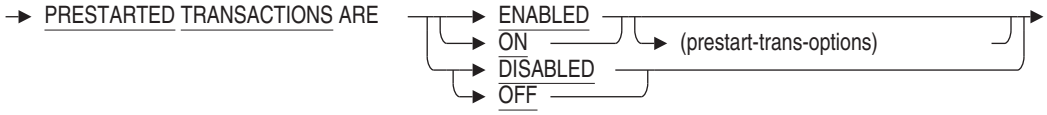
root-file-params-3 =



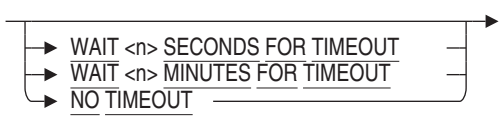
system-index-options =



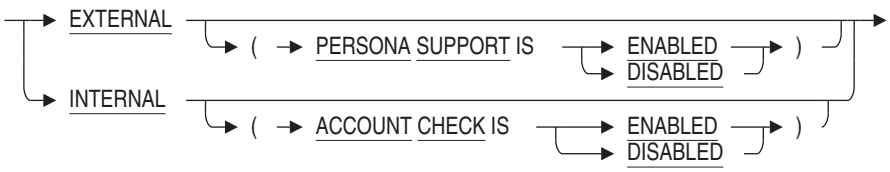
prestarted-transaction-params =



prestart-trans-options =

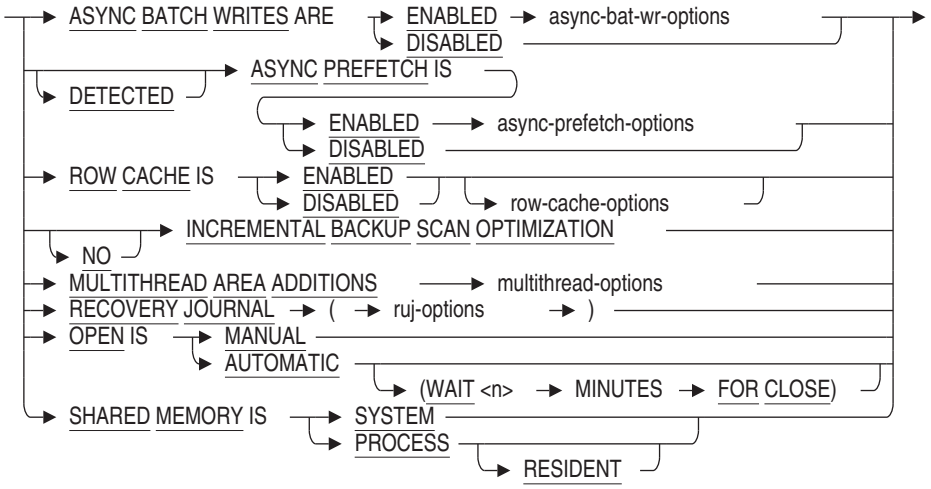


security-checking-options =

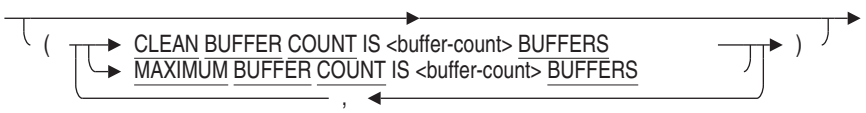


notify-options =

root-file-params-4 =

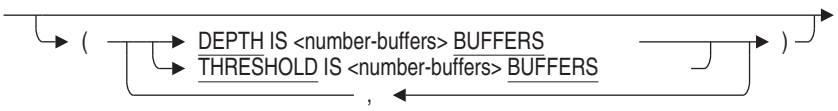


asynch-bat-wr-options =

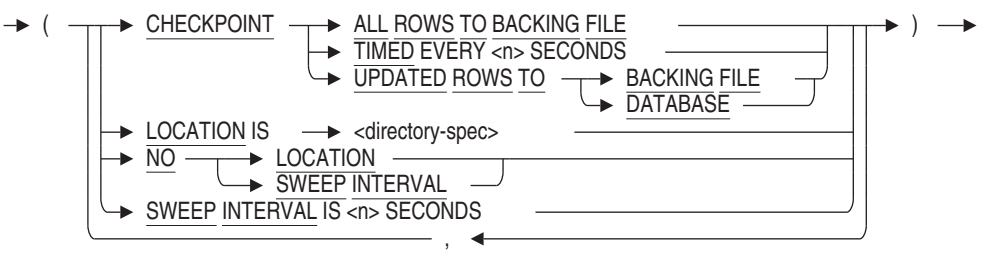


CREATE DATABASE Statement

async-prefetch-options =

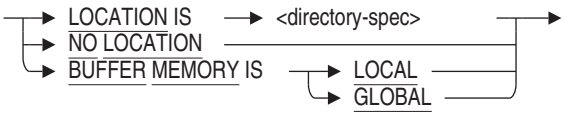


row-cache-options =

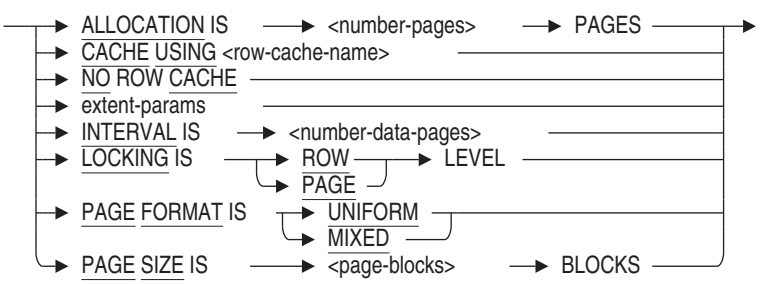


multithread-options =

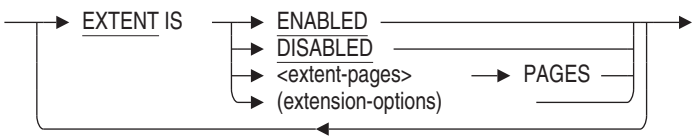
ruj-options =



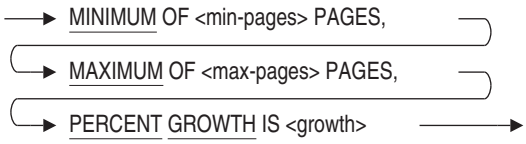
storage-area-params-1 =



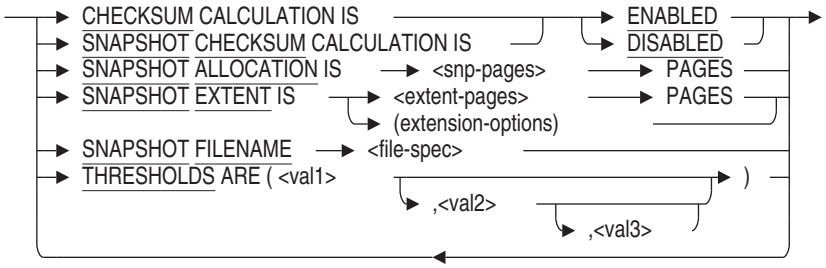
extent-params =



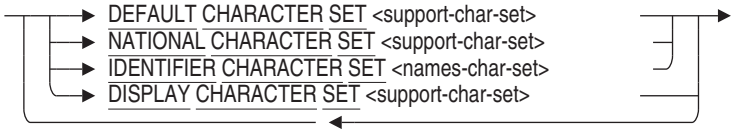
extension-options =

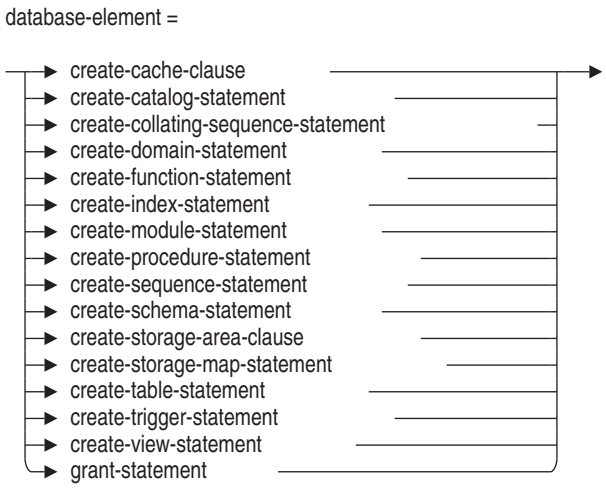


storage-area-params-2 =



character-sets =





Arguments

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.

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 2.2.1.

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.

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 700 pages. If you are loading a large database, a large allocation helps to prevent fragmented files.

ALTER COUNT RESULT
SET COUNT RESULT
By default, Rdb returns the result of the COUNT(*), COUNT (DISTINCT expression), and COUNT(expression) as a BIGINT type. The clause can be used to instruct SQL to fetch an INTEGER type result instead.

In most cases this change is unnecessary as any application that fetches a COUNT into an integer host variable will have the value automatically converted to the target data type. However, some applications expect an integer type (such as those written in Dynamic SQL) so this clause sets a permanent database attribute to ensure correct application behavior.

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.

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.

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.

BUFFER SIZE IS buffer-blocks BLOCKS
Specifies the number of blocks Rdb allocates per buffer. You need to specify an unsigned integer greater than zero and less than or equal to 256. The default buffer size is 4 times the PAGE SIZE value (16 blocks for the default PAGE SIZE of 4).

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.

CACHE USING row-cache-name
Assigns the named row cache 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 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 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. 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 database.

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.

CHECKPOINT TIMED EVERY n SECONDS
For the row-cache-options clause, specifies the frequency with which the row-cache server (RCS) process checkpoints the contents of the row caches back to disk. The RCS process does not use the checkpoint frequency options of the FAST COMMIT clause.

The frequency of RCS checkpointing is important in determining how much of an .aij file must be read during a recovery operation following a node failure. It also affects the frequency with which marked records get flushed back to the database for those row caches that checkpoint to the database. The default is every 15 minutes (900 seconds).

CHECKPOINT UPDATED ROWS TO BACKING FILE
CHECKPOINT UPDATED ROWS TO DATABASE
CHECKPOINT ALL ROWS TO BACKING FILE
Specifies the default source and target during checkpoint operations for all row caches. If ALL ROWS is specified, then the source records written during each checkpoint operation are both the modified and the unmodified rows in a row cache. If UPDATED ROWS is specified, then just the modified rows in a row cache are checkpointed each time.

If the target of the checkpoint operation is BACKING FILE, then the RCS process writes the source row cache entries to the backing (.rdc) files. The row cache LOCATION, ALLOCATION, and EXTENT clauses are used to create the backing files. Upon recovery from a node failure, the database recovery process is able to repopulate the row caches in memory from the rows found in the backing files.

If the target is DATABASE, then updated row cache entries are written back to the database. The row cache LOCATION, ALLOCATION, and EXTENT clauses are ignored. Upon recovery from a node failure, the database recovery process has no data on the contents of the row cache. Therefore, it does not repopulate the row caches in memory.

The CHECKPOINT clause of the CREATE CACHE, ADD CACHE, or ALTER CACHE clause overrides this database-level CHECKPOINT clause.

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 or snapshot files.

The default is ENABLED.

Note

Oracle Corporation 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 Corporation 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 Corporation 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.

CLEAN BUFFER COUNT IS buffer-count BUFFERS
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 one; 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. For information about how to set the values, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

COLLATING SEQUENCE collation-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.

Collation-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 (/).

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.

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 2.2.11.

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.

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-sequence-statement
See the CREATE SEQUENCE Statement for details.

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.

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.

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.

DBKEY SCOPE IS ATTACH
DBKEY SCOPE IS TRANSACTION
Controls when the database key of a deleted row can be used again by SQL. This setting is not a database root file parameter, but a characteristic of the implicit database attach 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.

- The DBKEY SCOPE IS ATTACH clause means that SQL cannot use the database key again (to refer to a newly inserted row) until all users who have attached with DBKEY SCOPE IS ATTACH have detached from the database.

  Also it only requires one process to attach with DBKEY SCOPE IS ATTACH to force all database users to assume this characteristic.

- Oracle Corporation recommends using DBKEY SCOPE IS TRANSACTION to prevent excessive consumption of storage area space by overhead needed to support DBKEY SCOPE IS ATTACH, and to prevent performance problems when storing new rows.

  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.

For more information, see Section 2.6.5.

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.

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$QUERY_OUTLINES
- RDB$SEQUENCES
- RDB$PROFILES
- RDB$GRANTED_PROFILES
- RDB$TYPES
- RDB$TYPE_FIELDS
- RDB$WORKLOAD
- RDB$OBJECT_SYNONYMS
- RDB$SYNONYMS
- RDB$CATALOG_SCHEMA

For information on moving these system tables to other storage areas, see the *Oracle Rdb Guide to Database Design and Definition*.

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.

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.

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 asynchronously fetches pages further in the sequence. This avoids wait times when the page is really needed.

Detected asynchronous prefetch is enabled by default.

DICTIONARY IS REQUIRED
DICTIONARY IS NOT REQUIRED
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.

DISPLAY CHARACTER SET
Specifies the character set encoding and characteristics expected of text strings returned back to SQL from Oracle Rdb. See the Usage Notes for additional information.

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 PAGES
EXTENT IS (extension-options)
Specifies the number of pages of each storage area file extent. For more information, see the SNAPSHOT EXTENT argument.

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
- 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.

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 file specification may contain an OpenVMS remote node specification. Oracle Rdb must be installed on that remote node so that the CREATE DATABASE statement can be executed remotely. Note that all other file specifications in the command (storage areas, snapshot files, recovery journal location, and so on) must be specified using the logical names and device names relative to that remote node.

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

  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.

FROM library-name
Specifies the name of an NCS library other than the default library. The default NCS library is SYS$LIBRARY:NCS$LIBRARY.

GALAXY SUPPORT IS ENABLED
GALAXY SUPPORT IS DISABLED
Allows global memory to be shared in an OpenVMS Galaxy configuration. Galaxy support is disabled by default.

OpenVMS Galaxy is a software architecture for the OpenVMS Alpha operating system that enables multiple instances of OpenVMS to execute cooperatively in a single computer. An instance refers to a copy of the OpenVMS Alpha operating system. As an extension of the existing OpenVMS cluster support within Oracle Rdb, Oracle Rdb provides support for databases opened on multiple instances (or nodes) within a Galaxy system to share data structures in memory. Within an Oracle Rdb Galaxy environment, all instances with an open database share:

- Database root objects (for example, TSN blocks and SEQ blocks)
- Global buffers (if enabled)
- Row caches and Row Cache Server process (RCS) (if enabled)

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 attach. For more than one attach 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 attach 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

If GALAXY SUPPORT is enabled, then a single global buffer pool is shared by all Galaxy nodes.

grant-statement
See the GRANT Statement for details.

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 Section 2.1.5 for a list of allowable character sets.

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.

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 Rdb Guide to Database Maintenance*.

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.

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;
```

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.

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 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. 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.

This clause is ignored if the row cache is defined to checkpoint to the database.

LOCK PARTITIONING IS ENABLED
LOCK PARTITIONING IS DISABLED
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.

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 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*.

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 and long transactions may use excessive locks.

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

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

LOGMINER SUPPORT IS ENABLED
LOGMINER SUPPORT IS DISABLED
Allows additional information to be written to the after-image journal file to allow the use of the RMU Unload After_Image command. See *Oracle RMU Reference Manual* for more details. Logminer support is disabled by default.

The LOGMINER SUPPORT clause allows the continuous mode for LogMiner to be enabled and disabled.

- LOGMINER SUPPORT IS ENABLED (CONTINUOUS)

  Enables continuous LogMiner.

- LOGMINER SUPPORT IS ENABLED (NOT CONTINUOUS)

  Disables continuous LogMiner, but leaves LogMiner enabled.

- LOGMINER SUPPORT IS DISABLED

  Disables LogMiner, including disabling continuous LogMiner.

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. For information about how to set the values, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

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

MEMORY ALLOCATION IS mem-octets
Allows the database administrator to define the size of the GLOBAL BUFFER pool using direct memory sizing as an alternate to specifying the NUMBER of pages. The value of mem-octets is an unsigned numeric literal or a quantified numeric literal. This clause is not compatible with the NUMBER IS clause. Use just one of the keywords to define the size of the global buffers.

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, TRUNCATE TABLE, COMMENT ON and RENAME 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.

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

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 2.2.18.

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.

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.

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.

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).

NO LOCATION
This is a subclause of other clauses and has different effects, depending upon the clause in which it is used, as follows:

- In the row-cache-options clause

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

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

- In a CREATE CACHE, ADD CACHE, or ALTER CACHE clause (row-cache-params1 clause)

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

  This clause is ignored if the row cache is defined to checkpoint to the database.

NO ROW CACHE
Specifies that the database default is not to assign a row cache 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 to override the database default. 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 database.

NOTIFY IS ENABLED
NOTIFY IS DISABLED
Specifies whether system notification is enabled or disabled.

When the system notification is enabled, the system is notified (using the OpenVMS OPCOM facility) in the event of events such as running out of disk space for a journal.

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 replaces any operator classes set by the RMU Set After_Journal Notify command.

The default is disabled.

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. 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.

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 524,288. The default is 250 buffers.

NUMBER OF CLUSTER NODES number-nodes (SINGLE INSTANCE)
NUMBER OF CLUSTER NODES number-nodes (MULTIPLE INSTANCE)
Sets the upper limit on the maximum number of VMS cluster nodes from which users can access the shared database. The default is 16 nodes. The range is 1 to 96 nodes. The actual maximum limit is the current VMS cluster node limit set by your system administrator.

The Oracle Rdb root file data structures (.rdb) are mapped to shared memory. Each such shared memory copy is known as an Rdb instance. When there is only one copy of shared memory containing root file information, several optimizations are enabled to reduce locking and root file I/O during database activity. To enable these optimizations, specify NUMBER OF CLUSTER NODES 1, or use the SINGLE INSTANCE clause.

MULTIPLE INSTANCE means that the Oracle Rdb root file data structures are mapped on different system and are kept consistent through disk I/O. Such systems can not benefit from single instance optimizations. MULTIPLE INSTANCE is the default.

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 524,288. The default value for the NUMBER OF RECOVERY BUFFERS parameter is 250 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.

You can override the default number of database recovery buffers by defining a value for the system logical name RDM$BIND_DBR_BUFFER_CNT. This will be used by database detached processes (DBR) when allocating buffers during recovery.

```
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 Rdb Guide to Database Maintenance*.

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 5 users.

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 CLUSTER NODES parameter, see the Usage Notes.

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.

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 4 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 4 blocks (2048 bytes). If your largest row is larger than free space on the page, 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.

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, or SINGLE INSTANCE must be specified in the NUMBER of CLUSTER NODES clause.
- 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*.

PATHNAME path-name

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.

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

PRESTARTED TRANSACTIONS ARE ENABLED
PRESTARTED TRANSACTIONS ARE DISABLED
Enables or disables the prestarting of transactions.

Note that the keyword ON, available in previous versions, is synonymous with ENABLED, and the OFF keyword is synonymous with the DISABLED keyword.

This clause is used to establish a permanent database setting for prestarted transactions. In prior versions, this clause was only used to temporarily set the mode for prestarted transaction for the implicit attach performed by the CREATE DATABASE and IMPORT DATABASE statements.

The prestart-trans-options can be one of the following clauses:

- WAIT n SECONDS FOR TIMEOUT

  The n represents the number of seconds to wait before aborting the prestarted transaction. Timing out the prestarted transaction may prevent snapshot file growth in environments where servers stay attached to the database with long periods of inactivity.

- WAIT n MINUTES FOR TIMEOUT

  The n represents the number of minutes to wait before aborting the prestarted transaction.

- NO TIMEOUT

  This is the default for a prestarted transaction.

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 or database object; matches any held roles (role rights identifiers); gets whatever privileges have been granted to those roles; and gets the privileges defined for PUBLIC. A user without an entry in the access privilege set and who holds no matching roles 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.

RECOVERY JOURNAL (BUFFER MEMORY IS LOCAL)
RECOVERY JOURNAL (BUFFER MEMORY IS GLOBAL)
Specifies whether RUJ buffers will be allocated in global or local memory.

The RUJ buffers used by each process are normally allocated in local virtual memory. With the introduction of row caching, these buffers now can be assigned to a shared global section (global memory) on OpenVMS, so that the recovery process can process this in-memory buffer and possibly avoid a disk access.

You can define this buffer memory to be global to improve row caching performance for recovery. If row caching is disabled, then buffer memory is always local.

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.

If this clause is omitted, then NO LOCATION is assumed.

Following is an example using this clause:

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

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

RECOVERY JOURNAL (NO LOCATION)

If you specify NO LOCATION, the recovery journal uses the current user's login device and the directory [RDM$RUJ]. See the *Oracle Rdb Guide to Database Maintenance* for more information on recovery-unit journal files.

RESERVE n CACHE SLOTS

Specifies the number of row caches for which slots are reserved in the database.

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. You can only add row caches if row cache slots are 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 caches 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 caches

If you do not specify the RESERVE CACHE SLOTS clause, the default number of row caches is one.

Reserving row cache slots is an offline operation (requiring exclusive database access). See the CREATE CACHE Clause for more information.

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 SEQUENCES

Specifies the number of sequences for which slots are reserved in the database. Sequences are reserved in multiples of 32. Thus, if you specify a value less than 32 for n, 32 slots are reserved. If you specify a value of 33, 64 slots are reserved, and so on.

You can use the RESERVE SEQUENCES clause to reserve slots in the database root file for future use by the CREATE SEQUENCE statement. Sequences can be created only if sequence slots are available. Slots become available after a DROP SEQUENCE statement or a RESERVE SEQUENCES clause of the ALTER DATABASE statement is executed.

The number of reserved slots for sequences cannot be decreased.

If you do not specify the RESERVED SEQUENCES clause, the default number of sequence slots is 32.

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.

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.

root-file-params-1
root-file-params-2
root-file-params-3
root-file-params-4
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.

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 is created and assigned to one or more storage areas.

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

The following conditions must be true in order to use row caches:

- The number of cluster nodes is one
- After-image journaling is enabled
- Fast commit is enabled
- One or more cache slots are reserved
- Row caching is enabled

Use the RMU Dump Header command to check if you have met the requirements for using row caches. The following command output displays a warning for every requirement that is not met:

```
   .
   .
   .
Row Caches...
- Active row cache count is 0
- Reserved row cache count is 1
- Sweep interval is 1 second
- Default cache file directory is ""
- WARNING: Maximum node count is 16 instead of 1
- WARNING: After-image journaling is disabled
- WARNING: Fast commit is disabled
   .
   .
   .
```

ROWID SCOPE IS ATTACH
ROWID SCOPE IS TRANSACTION

The ROWID keyword is a synonym for the DBKEY keyword. See the DBKEY SCOPE IS argument for more information.

SECURITY CHECKING

Traditionally, Oracle Rdb has performed security checking using the operating system security layer (for example, the UIC and rights identifiers of the OpenVMS operating system).

The access control list (ACL) information stored in the database contains a granted privilege mask and a set of users represented by a unique integer (for example, a UIC).

There are two modes of security checking:

1. SECURITY CHECKING IS EXTERNAL

   This is the default. External security checking recognizes database users (created with the SQL CREATE USER statement) as operating system user identification codes (UICs) and roles as special rights identifiers or groups. PERSONA support is enabled or disabled as follows:

   - SECURITY CHECKING IS EXTERNAL (PERSONA SUPPORT IS ENABLED)

Enables the full impersonation of an OpenVMS user. This means the UIC and the granted right identifiers are used to check access control list permissions. This is the default.

- SECURITY CHECKING IS EXTERNAL (PERSONA SUPPORT IS DISABLED)

  Disables the full impersonation of an OpenVMS user. Only the UIC is used to check access control list permissions.

2. SECURITY CHECKING IS INTERNAL

   In this mode, Oracle Rdb records users (username and UIC) and roles (rights identifiers) in the database. The CREATE USER and CREATE ROLE statements perform this action explicitly, and GRANT will perform this implicitly. This type of database can now be moved to another system and is only dependent on the names of the users and roles.

   - SECURITY CHECKING IS INTERNAL (ACCOUNT CHECK IS ENABLED)

     The ACCOUNT CHECK clause ensures that Oracle Rdb validates the current database user with the user name (such as defined with an SQL CREATE USER statement) stored in the database. This prevents different users with the same name from accessing the database. Therefore, this clause might prevent a breach in security.

     The ACCOUNT CHECK IS ENABLED clause on OpenVMS forces the user session to have the same user name and UIC as recorded in the database.

   - SECURITY CHECKING IS INTERNAL (ACCOUNT CHECK IS DISABLED)

     If you specify the ACCOUNT CHECK IS DISABLED clause, then a user with a matching UIC (also called a profile-id) is considered the same as the user even if his or her user name is different. This allows support for multiple OpenVMS users with the same UIC.

SEGMENTED STRING STORAGE AREA IS area-name
Another name for LIST STORAGE AREA.

SET TRANSACTION MODES
Enables only the modes specified, disabling all other defaulted 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.

SHARED MEMORY IS PROCESS RESIDENT
The SHARED MEMORY clause determines whether database root global sections (including global buffers when enabled) or whether the cache global sections are created in system space or process space. The RESIDENT option extends the PROCESS option by making the global section memory resident.

SHARED MEMORY IS SYSTEM
SHARED MEMORY IS PROCESS
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 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.

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 CHECKSUM ALLOCATION IS ENABLED
SNAPSHOT CHECKSUM ALLOCATION IS DISABLED
See the CHECKSUM ALLOCATION clause for details.

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 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, directory, and file name as the database root file.

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.

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.

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. For more information about when to use statistics collection, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

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.

SWEEP INTERVAL IS n SECONDS

Specifies the interval, in seconds, between each Record Cache Server (RCS) sweep. Allowable values must be in the range from 1 second to 3600 seconds (1 hour). The default is 1.

The **Record Cache Server (RCS)** is a detached server process automatically invoked by the monitor when row caching is active.

A **sweep** is one full pass through all active row cache areas to write modified rows back to the database storage areas.

SYSTEM INDEX (COMPRESSION IS . . .)

This clause allows the database creator to choose compressed system indexes. The default is SYSTEM INDEX (COMPRESSION IS ENABLED).

If enabled 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 non-character 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 IS . . .) clause, you only can change it using the EXPORT and IMPORT statements. You cannot alter the database to change the compression mode.

The clause SYSTEM INDEX COMPRESSION IS is identical to this clause and is retained for compatibility with older versions of Oracle Rdb.

SYSTEM INDEX (PREFIX CARDINALITY COLLECTION IS . . .)

This clause allows the database creator to adjust the prefix cardinality collection for system indices. Refer to the CREATE INDEX Statement for more details on these clauses. The default is PREFIX CARDINALITY COLLECTION IS ENABLED.

SYSTEM INDEX (TYPE IS . . .)

This clause allows the database creator to choose between SORTED or SORTED RANKED indices for system table. The default is SORTED RANKED. SORTED RANKED indices have advantages in space usage and reduced CPU during DDL operations for those system indices with many duplicates.

SYNONYMS ARE ENABLED
Adds the optional system table RDB$OBJECT_SYNONYMS that is used for the CREATE SYNONYM, ALTER . . . RENAME TO and RENAME statements. The default if omitted is disabled.

THRESHOLD IS number-buffers BUFFERS
This number represents the number of sequential buffer accesses that must be detected before prefetching is started. The default is four buffers.

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.

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 Rdb Guide to Database Maintenance*.

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.

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$BIND_BUFFERS logical name to use all the buffers in the global buffer pool. (The behavior of RDM$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.

USING 'password'
USING ('password', 'password')
A character string literal that specifies the user's password for the user name specified in the USER clause. If the user requires two passwords, then specify both values in parenthesis and separated by a comma.

txn-modes
Specifies the transaction modes for the database.

Mode	Description
ALL	All modes are enabled.
NONE	No modes are enabled.

Mode	Description
	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, 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 tables to be reserved for shared mode. That is, other users can work with those tables.
[NO] PROTECTED [READ | WRITE]	Allows tables to be reserved for protected mode. That is, other users can read from those tables.
[NO] EXCLUSIVE [READ | WRITE]	Allows tables to be reserved for exclusive access. That is, other users are prevented access to those tables, even in READ ONLY transactions.
ALL	Allows other users to work with all tables.
NONE	Allows no access to tables.

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

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.

Usage Notes

- The CREATE DATABASE statement starts and commits several transactions. However, 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.
- 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 Rdb 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 privileges to the database and no 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 Rdb 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.
- 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.

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*.

- 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 2.1.7.

- 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 2.1.5.
- 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 MIA, SQL92, SQL99 or SQL2011. 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 MIA, SQL92, SQL99 or SQL2011. 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 MIA, SQL92, SQL99 or SQL2011. Otherwise, the identifier character set is DEC_MCS if this clause is omitted.
- Specifying the DISPLAY CHARACTER SET clause on the ATTACH, CREATE DATABASE, or DECLARE ALIAS statements takes precedence over any previously issued SET DISPLAY CHARACTER SET or SET AUTOMATIC TRANSLATION statements.
- If the database default character set is not DEC_MCS, the PATHNAME specifier cannot be used due to limitations 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.
- 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.
- Oracle Corporation 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.
- 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.
- The RDB$PROFILES system table is used to record users, roles and profiles created with the CREATE USER, CREATE PROFILE, and CREATE ROLE statements. When a database is created, the creator is automatically added as a user.
- The GRANT statement may reference operating system users or groups prior to those users or roles being created in the database. In this case, Oracle Rdb automatically creates users and roles that correspond to the IDENTIFIED EXTERNALLY clause of the CREATE USER or CREATE ROLE statements.
- A node specification may only be specified for the root FILENAME clause of the CREATE DATABASE statement.

  This means that the directory or file specification specified with the following clauses can only be a device, directory, file name, and file type:

  – LOCATION clause of the ROW CACHE IS ENABLED, RECOVERY JOURNAL, ADD CACHE, and CREATE CACHE clauses
  – SNAPSHOT FILENAME clause
  – FILENAME and SNAPSHOT FILENAME clauses of the ADD STORAGE AREA and CREATE STORAGE AREA clauses
- Very large memory (VLM) allows Oracle Rdb to use as much physical memory as is available on your system and to dynamically map it to the virtual address space of database users. VLM provides access to a large amount of physical memory through small virtual address windows. Even though VLM is defined in physical memory, the virtual address windows are defined and maintained in each user's private virtual address space. Global buffers in VLM are fully resident, or pinned, in memory and do not directly affect the quotas of the working set of a process.

  The number of virtual address "windows" per process is based on the global buffers maximum 'allocate set' parameter specified with the USER LIMIT IS value of the ALTER DATABASE . . . GLOBAL BUFFERS command and is not directly adjustable.

The LARGE MEMORY parameter of the ALTER DATABASE . . . GLOBAL BUFFERS command is used to specify that global buffers are to be created in very large memory:

```
SQL> ALTER DATABASE FILENAME 'MYBIGDB.RDB'
cont> GLOBAL BUFFERS ARE ENABLED
cont>   (NUMBER IS 250000,
cont>    USER LIMIT IS 500,
cont>    LARGE MEMORY IS ENABLED);
```

It is important that you consider the amount of memory available on your system before you start using VLM for global buffers. You can use the DCL command SHOW MEMORY/PHYSICAL to check the availability and usage of physical memory. This command displays information on how much memory is used and how much is free. The free memory is available for VLM global buffers in addition to user applications.

The total number of global buffers per database is limited to 524,288 and the maximum buffer size is 64 blocks. This yields a global buffer maximum of 16gb (2,097,152 Alpha pages). This restriction may be relaxed in future releases of Oracle Rdb.

Refer to the *Oracle Rdb7 Guide to Database Performance and Tuning* for additional information about the global buffers feature.

- To enable or disable SHARED MEMORY IS PROCESS RESIDENT or LARGE MEMORY the process executing the command must be granted the VMS$MEM_RESIDENT_USER rights identifier. When this feature is enabled then the process that opens the database must also be granted the VMS$MEM_RESIDENT_USER identifier. Oracle recommends that the RMU/OPEN command be used when utilizing this feature.

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>     SNAPSHOT ALLOCATION IS 1 PAGES;
```

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 'SQL99';
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
Display character set is UNSPECIFIED

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 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
     Reserving Slots for Sequences
    No Sweep Thresholds
        Allocation:            100 blocks
        Extent:                100 blocks
```

Example 9: Reserving Slots for Sequences

```
SQL> CREATE DATABASE FILENAME many_sequences
cont> RESERVE 320 SEQUENCES;
```

Example 10: Creating a Database with a Row Cache

```
SQL> create database
cont>     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, type sorted ranked)
cont>     row cache is enabled
cont>     workload collection is enabled
cont>     restricted access
cont>     default storage area is AREA1
cont>     open is automatic (wait 5 minutes for close)
cont>
cont>     create cache CACHE_AREA1
cont>         shared memory is process
cont>         row length is 1000 bytes
cont>         cache size is 204 rows
cont>         checkpoint all rows to backing file
cont>
cont>     create storage area AREA1
cont>         page format is UNIFORM
cont>         cache using CACHE_AREA1
cont> ;
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 ENABLED
        Unused Storage Areas:         10
        Unused Journals:              10
        Unused Cache Slots:           5
        Unused Sequences:             32
        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
        Workload Collection is Enabled
        Cardinality Collection is Enabled
        Metadata Changes are Enabled
        Row Cache is Enabled
        Row cache: No Location
        Row cache: checkpoint updated rows to backing file
        Detected Asynch Prefetch is Enabled (depth is 4, threshold is 4)
        Default Storage Area AREA1
        Mode is Open Automatic (Wait 5 minutes for close)
        No RUJ File Location
        recovery journal buffers are in local memory
        Database Transaction Mode(s) Enabled:
            ALL
        Shared Memory:         Process
        Large Memory:          Disabled
        Security Checking is External
        System Index Compression is ENABLED
        System Index:
            Type is sorted ranked
            Prefix cardinality collection is enabled
        Logminer support is disabled
        Galaxy support is disabled
        Prestarted transactions are enabled
        Dictionary Not Required
        ACL based protections
Storage Areas in database with filename SAMPLE
     AREA1                          Default storage area
     RDB$SYSTEM                     List storage area.
Journals in database with filename SAMPLE
     No Journals Found
Cache Objects in database with filename SAMPLE
     CACHE_AREA1
```

Example 11: Using the MEMORY ALLOCATION clause when creating global buffers

```
create database
    filename TEST_DB

    allocation 110k pages
    snapshot allocation 1k pages
    global buffers are disabled (memory allocation 1m)

    create storage area AREA1
        allocation 100k pages
        snapshot allocation 2k pages
;
```

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.

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
  - Formatting clauses such as EDIT STRING or QUERY HEADER
- 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.

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 functions and procedures. If the domain has to change, you need only change that one domain definition (using the ALTER DOMAIN statement) to change all the tables. This ability makes it easier to keep applications consistent.

A domain can be referenced in the following locations:

- CREATE, ALTER and DROP DOMAIN statements
- CREATE and ALTER TABLE statements as the data type for a column
- CREATE and ALTER MODULE statements as the data type of a routine parameter, or the data type of declared variable
- CREATE FUNCTION statement as the data type of a function parameter or function result

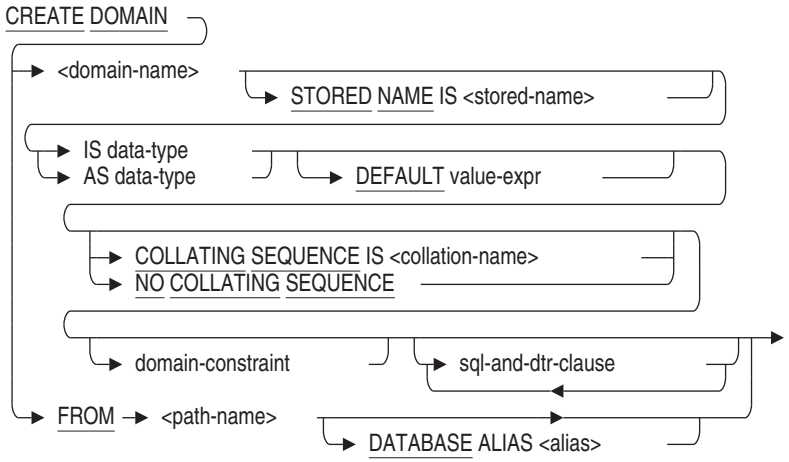
- CREATE PROCEDURE statement as the data type of a procedure parameter
- CREATE and ALTER SYNONYM statement as the base object for a synonym
- as the datatype of a CAST expression
- as a data type of a DECLARE variable statement in interactive SQL
- as the source in the EDIT USING clause of the SELECT and PRINT statements in interactive SQL. The EDIT STRING is inherited from that domain.

Environment

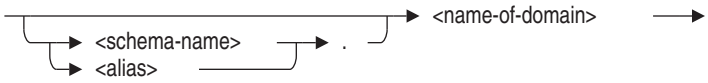
You can use the CREATE DOMAIN statement:

- 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

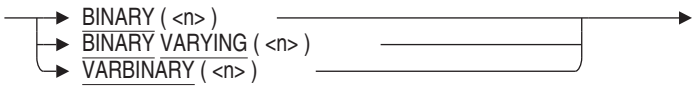
Format



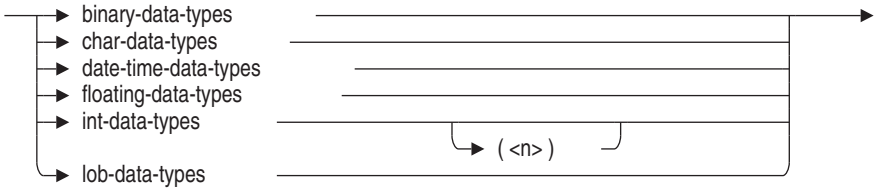
domain-name =



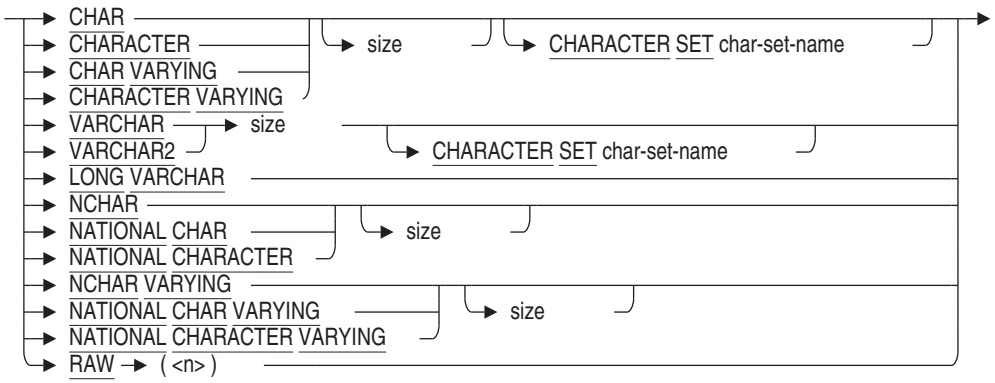
binary-data-types =



data-type =

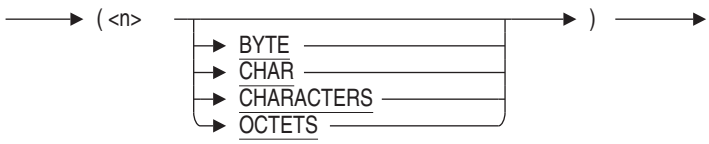


char-data-types =

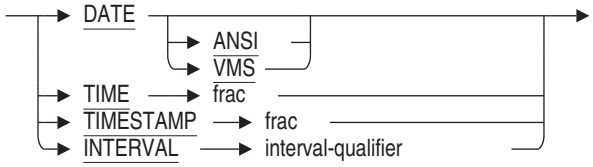


CREATE DOMAIN Statement

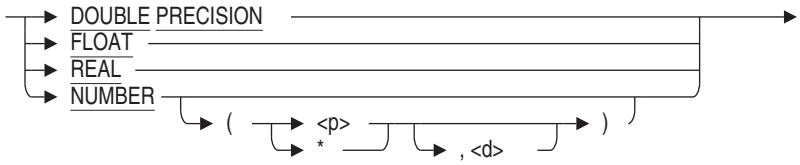
size =



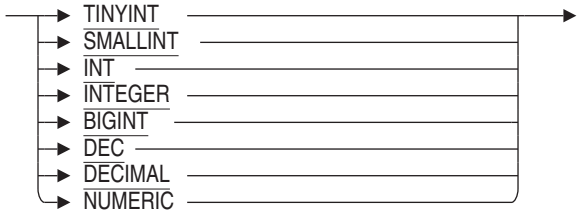
date-time-data-types =



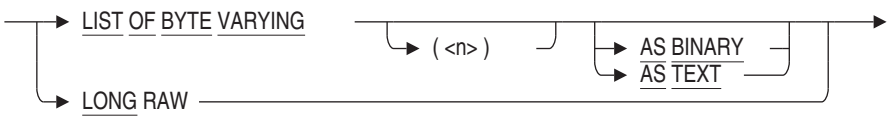
floating-data-types =



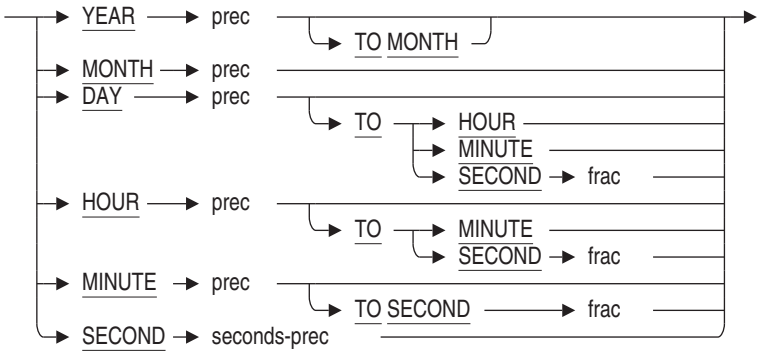
int-data-types =



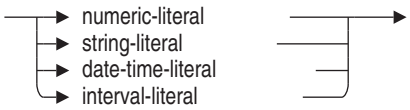
lob-data-types =



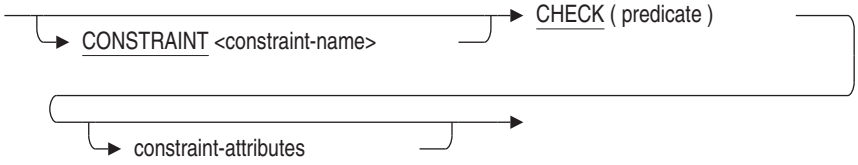
interval-qualifier =



literal =

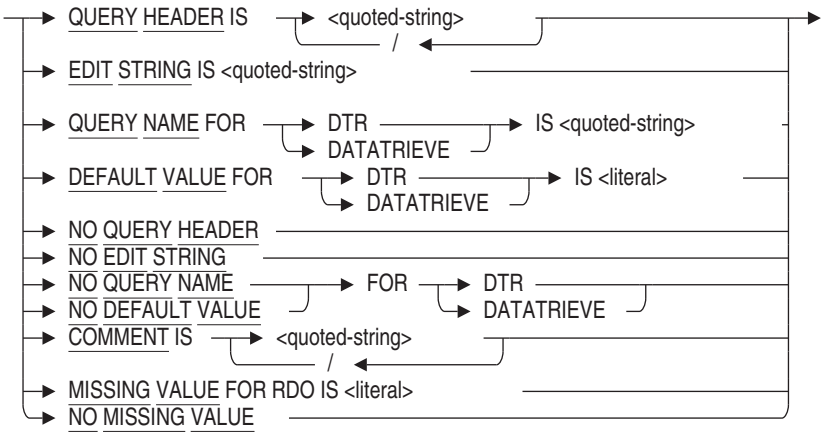


domain-constraint =



constraint-attributes =

sql-and-dtr-clause =



Arguments

char-data-types
A character type. See Section 2.3 for more information on data types.

character-set-name
A valid character set.

COLLATING SEQUENCE IS collation-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.

COMMENT IS 'string'
Adds a comment about the domain. SQL displays the text of the comment when it executes a SHOW DOMAIN statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

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 2.2.1 for more information on default databases and aliases.

date-time-data-types
A data type that specifies a date, time, or interval. See Section 2.3.2 for more information about date-time data types.

DEFAULT value-expr
Provides a default value for a domain.

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.

For more information about NULL, see Section 2.6.1 and the Usage Notes following this Arguments list.

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.

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

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.

A name can be specified for a domain CHECK constraint. The name can later be used by the ALTER DOMAIN .. DROP CONSTRAINT statement. This name must not be the same as an existing domain, table or column constraint nor that of a view WITH CHECK OPTION. The CONSTRAINT name is a simple identifier. It cannot be qualified by an alias as it is not a separate database object.

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.

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.

IS data-type
AS data-type
A valid SQL data type. See Section 2.3 for more information on data types.

MISSING VALUE FOR RDO IS literal
NO MISSING VALUE
The MISSING VALUE defines a string that is returned when an RDBPRE application fetches a NULL column value. The literal specified for the MISSING VALUE must be compatible with the data type of the domain. Only simple literal expressions are supported; namely DATE VMS literals, character string literals and numeric values (fixed or floating).

NO MISSING VALUE will remove any MISSING VALUE for the domain.

This clause is provided in SQL to allow applications written using the RDBPRE precompilers and databases created using the RDO database definition language to be migrated to SQL-only interfaces and still retain this functionality.

Oracle does not recommend the use of MISSING VALUE for modern applications because these semantics are not defined by the ANSI nor ISO SQL Database Language Standards. MISSING VALUE was introduced for RDO as a way to manage NULL (unknown) values in the database. SQL supports well defined standard functions IS NULL, NULLIF, and COALESCE for such purposes, as well as allowing applications to fetch the null indication into an INDICATOR variable.

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.

sql-and-dtr-clause
Optional SQL and DATATRIEVE formatting clause. See Section 2.5 for more information on formatting clauses.

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 2.2.18.

Usage Notes

- 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 formatting 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.
- The data type of a value specified in the DEFAULT 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' ;
```

- You might not want to use domains when you create tables if:
  - Your application must be compatible with Oracle Database.
  - 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.
- 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 explicitly specify the length of the data type in characters or octets. By default, lengths are specified in octets. By preceding the CREATE DOMAIN statement with the SET CHARACTER LENGTH 'CHARACTERS' or SET DIALECT to a modern dialect, you change the length units to characters. For more information, see the SET CHARACTER LENGTH Statement and the SET DIALECT Statement.
- If you create a character type domain without specifying a character set then it will be assigned with the database default character set.

- 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.
- 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 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 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;
```

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:

```
$ 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                     BIGINT
```

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 'SQL99' statement sets the default date format
SQL> -- to the ANSI/ISO SQL standard format.
SQL> --
SQL>  SET DIALECT 'SQL99';
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 smallint or integer numeric columns
- Compression of space characters from text data types and of binary zeros from nontext data types
- Thresholds for the logical storage areas that contain the index
- A comment for the index definition
- Whether logging to the .ruj and .aij files is enabled or disabled for the create index operation

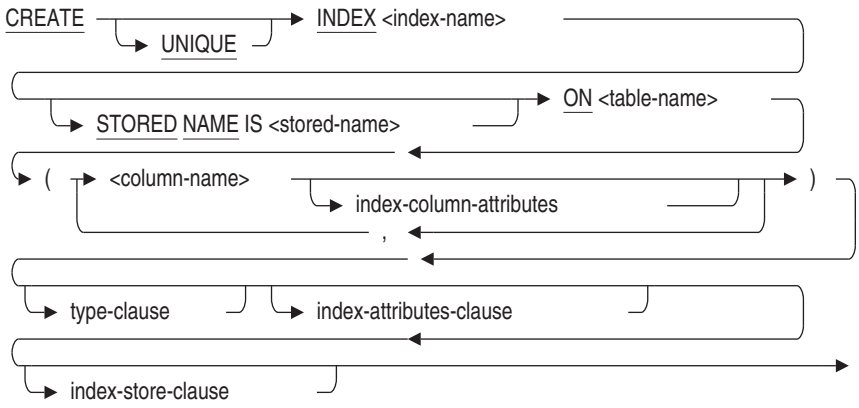
Environment

You can use the CREATE INDEX statement:

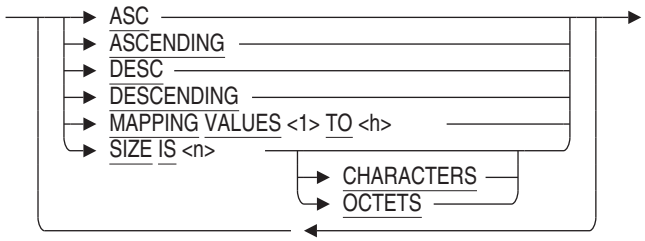
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

CREATE INDEX Statement

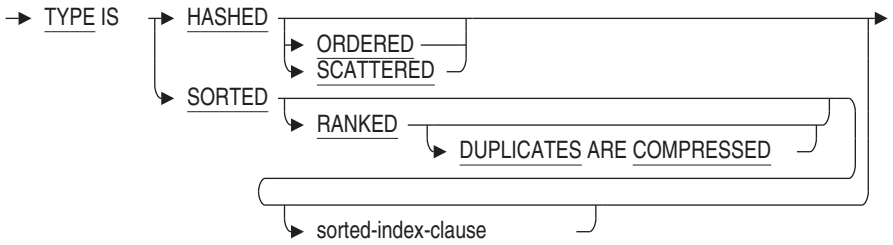
Format



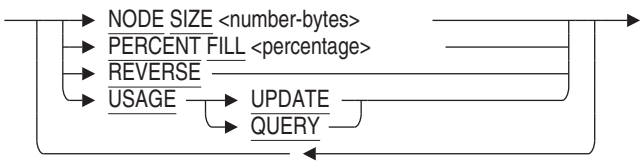
index-column-attributes =



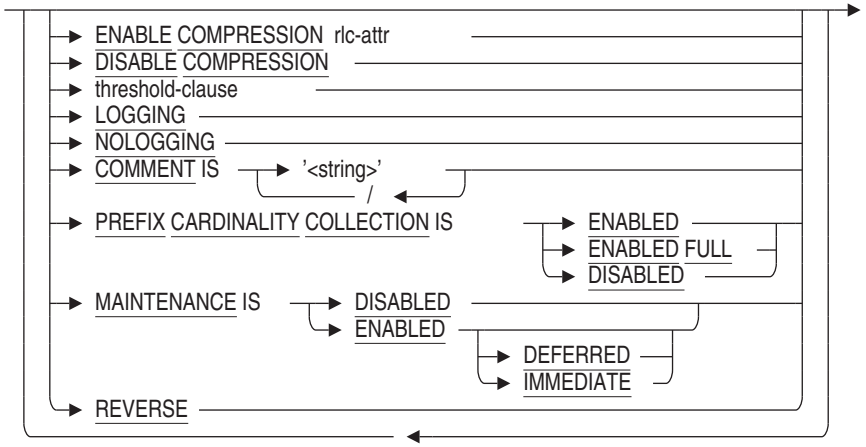
type-clause =



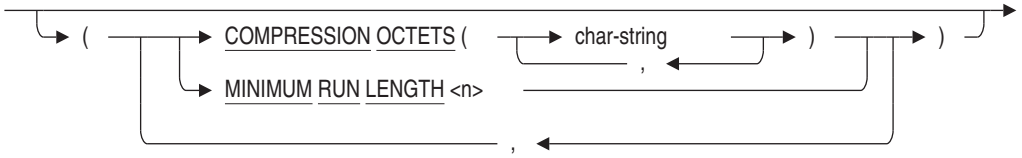
sorted-index-clause =



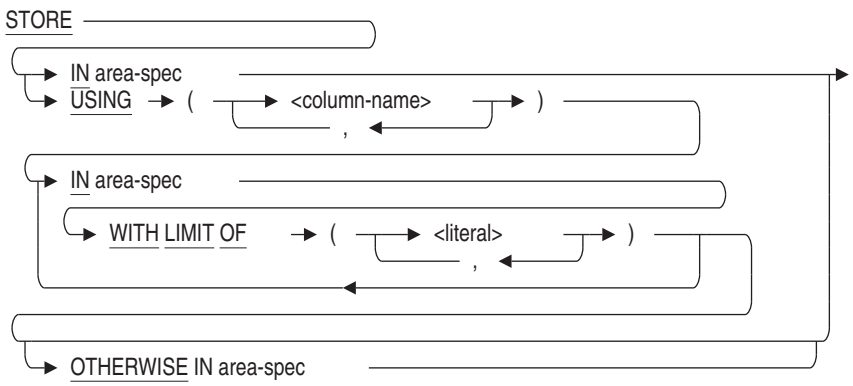
index-attributes-clause =



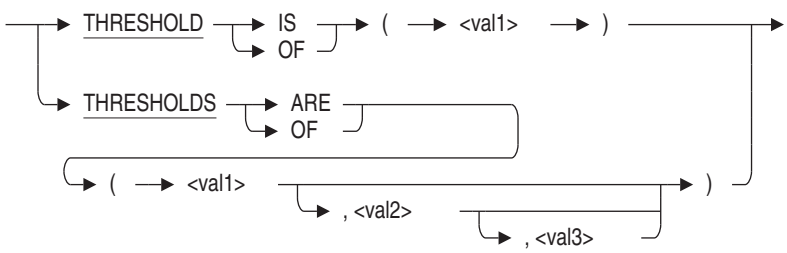
rlc-attr =



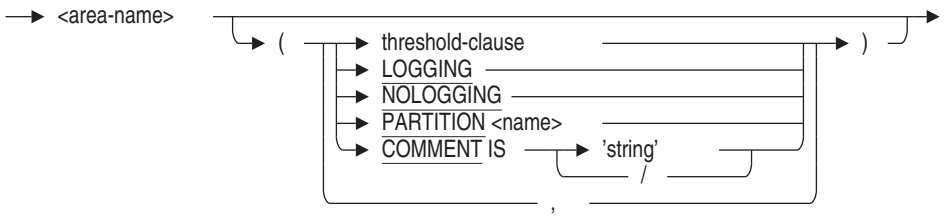
index-store-clause =



threshold-clause =



area-spec =



Arguments

ASC

ASCENDING
An optional keyword that causes SQL to create ascending index segments. If you omit the ASC, ASCENDING, DESC or DESCENDING keyword, ascending is the default.

BUILD ALL PARTITIONS
This clause operates on an index in build-pending state (created using MAINTENANCE IS ENABLED DEFERRED) and builds all incomplete partitions. If the index is not in build-pending state then the statement completes successfully with a warning.

No other clauses may appear in the same ALTER INDEX statement.

BUILD PARTITION partition-name
This clause operates on an index in build-pending state (created using MAINTENANCE IS ENABLED DEFERRED) and builds the named partition. If the index is not in build-pending state then the statement completes successfully with a warning.

No other clauses may appear in the same ALTER INDEX statement.

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 CHAR, VARCHAR or LONG VARCHAR data type, the size of the column must be less than or equal to 254 characters, or the SIZE IS clause must be used.

COMMENT IS 'string'
Adds a comment about the storage map definition for the index. SQL displays the text of the comment when it executes a SHOW INDEXES statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

COMPRESSION OCTETS
Allows user specification of the octets used for run length compression.

The COMPRESSION OCTETS clause can be used to specify those characters that will be used for run-length compression. If this clause is not defined, then Rdb will use the space character of the character set of any CHAR or VARCHAR columns in the index, and a zero for any other data type.

DESC
DESCENDING
An optional keyword that causes SQL to create descending index segments. If you omit the ASC, ASCENDING, DESC or DESCENDING keywords, ascending is the default.

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.

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 Rdb Guide to Database Design and Definition* for more information on sorted ranked B-tree indexes.

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 Rdb Guide to Database Design and Definition*.

IN area-name
Associates the index directly with a single storage area. All entries in the index are stored in the area you specify.

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.

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 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 default 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.

LOGGING
NOLOGGING
The LOGGING clause specifies that index nodes and hash buckets be logged when written to the database. Logging includes writing data and management records to the recovery-unit journal file (.ruj) and after-image journal files (.aij). When the NOLOGGING clause is specified, then only a small number of management records are logged in the recovery-unit journal file (.ruj) and after-image journal files (.aij). See the Usage Notes below for more information.

LOGGING and NOLOGGING can be specified per storage area (partition) or as a default for the CREATE INDEX statement. The LOGGING and NOLOGGING clauses are mutually exclusive; specify only one. The LOGGING clause is the default.

MAINTENANCE IS DISABLED
An index created using this clause is not maintained. The index definition serves only as a template.

MAINTENANCE IS ENABLED DEFERRED
An index created using this clause does not contain index keys for the current rows in the table. Until this index is built (using ALTER INDEX . . . BUILD), the index is placed in a build-pending state. Any table with a build-pending index can not be updated using the INSERT, DELETE, or UPDATE statements.

MAINTENANCE IS ENABLED IMMEDIATE
This is the default behavior for CREATE INDEX.

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 SMALLINT or INTEGER. 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 $2^{31} - 4 \times 10^{scale}$

  If the value of the key is less than zero or greater than $2^{31} - 4 \times 10^{scale}$, Oracle Rdb signals an exception.

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.

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 size is computed based on the free space on the target storage area page. The exception is the case of SORTED indices allowing duplicates where the size will be:

- 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 length* + *overhead*) + 32) $\leq$ *node size* $\leq$ 32767

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.

PARTITION name
Names the partition. The name can be a delimited identifier. Partition names must be unique within the index. If you do not specify this clause, Oracle Rdb generates a default name for the partition.

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 85 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.

PREFIX CARDINALITY COLLECTION IS DISABLED
This setting disables the cardinality collection and, instead, uses a fixed scaling algorithm which assumes a well balanced index tree.

PREFIX CARDINALITY COLLECTION IS ENABLED
This is the default behavior for CREATE INDEX. The Oracle Rdb optimizer collects approximate cardinality values for the index columns to help in future query optimization. Note that no extra I/O is incurred to collect these values and, therefore, adjacent key values from other index nodes can not be checked. Hence, some inaccuracy may be seen for these indexes. In most cases, this is adequate for query optimizations.

PREFIX CARDINALITY COLLECTION IS ENABLED FULL
This setting requests that extra I/O be performed, if required, to ensure that the cardinality values reflect the key value changes of adjacent index nodes.

REVERSE
SORTED and SORTED RANKED indices created with the REVERSE attribute have the index keys encoded so that the octet and bit order is reversed. Conceptually, this means that adjacent values are changed so that they are no longer consecutive. This key reversal attempts to spread index access to different nodes for normally adjacent values and is best when applied to automatically generated key values for high insert environments.

When using the REVERSE attribute for the index, the column ordering clauses ASC and DESC are not permitted. REVERSE cannot be applied to a HASHED index. The STORE USING clause is not allowed for REVERSE indices.

REBUILD ALL PARTITIONS
This clause combines the TRUNCATE and BUILD actions into a single function. No other clauses may appear in the same ALTER INDEX statement.

REBUILD PARTITION partition-name
This clause combines the TRUNCATE and BUILD actions into a single function for the named partition. No other clauses may appear in the same ALTER INDEX statement.

SIZE IS n [CHARACTERS | OCTETS]
A compression clause for character or character varying (varchar) index keys that limits the number of characters used for retrieving data. The *n* specifies the number of characters or octets of the key that are used in the index. An optional CHARACTERS or OCTETS keyword can be used to specify the units used for the size. Otherwise, the current character length setting for the module or session will be used.

Note

Although you can create a SIZE IS index and specify the UNIQUE clause, truncating the index key values may make the key values non-unique. In this case, the index definition or insert or update statements fail.

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.

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 only include a subset of the columns, you must include the leading segments of the multisegmented index.

For example, if a CREATE INDEX statement specifies a multisegmented index based on the columns LAST_NAME, FIRST_NAME, and MIDDLE_INITIAL, then the USING clause must include the first segment LAST_NAME, or the first two segments, LAST_NAME, and FIRST_NAME, or all the segments of the index. This is true for sorted indices only.

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.

There is no restriction for hashed scattered indexes. For hashed ordered indexes, all segments listed, except the last segment can be included. Also, HASHED ORDERED indexes have further restrictions on the data type of the final column; it must be a date/time or integer numeric data type.

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 2.2.18.

table-name
The name of the table that includes the index. The table must be in the same schema as the index.

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 Rdb Guide to Database Design and Definition*.

TRUNCATE ALL PARTITIONS

This clause operates in a similar way to TRUNCATE TABLE, but just on one index. The index is automatically set to MAINTENANCE IS ENABLED DEFERRED (i.e. build-pending state) if it was currently ENABLED IMMEDIATE. Otherwise is stays in a disabled state.

No other clauses may appear in the same ALTER INDEX statement.

TRUNCATE PARTITION partition-name

This clause operates on just the named index partition. The index is automatically set to MAINTENANCE IS ENABLED DEFERRED (that is, build-pending state) if it was currently ENABLED IMMEDIATE. Otherwise is stays in a disabled state.

No other clauses may appear in the same ALTER INDEX statement.

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 hashing 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: [1]

[1] All 64 bits are used when these data types are used.

- The last column of the index key must be one of the following data types:
  - TINYINT
  - SMALLINT
  - INTEGER
  - BIGINT [1]
  - DATE (both ANSI and VMS) [1]
  - TIME [1]
  - TIMESTAMP [1]
  - INTERVAL [1]
- 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 hashing algorithm to the index key. As a result, the distribution pattern is guaranteed to follow the index key distribution. 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 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 Rdb 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 (>)). (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 may use 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 Rdb 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.

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.

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.

USAGE UPDATE
USAGE QUERY
Specifies a PERCENT FILL value appropriate for update-intensive or query-intensive applications. The USAGE UPDATE clause sets the PERCENT FILL value at 85 percent. The USAGE QUERY clause sets the PERCENT FILL value at 100 percent.

You should specify either the PERCENT FILL or USAGE clause, but not both. If neither PERCENT FILL nor USAGE clauses are specified, then the default will be 85 percent.

You cannot specify this argument in an CREATE INDEX statement that refers to a HASHED index.

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 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.

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 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.
- 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
```

  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.

  For more information on this restriction, see the *Oracle Rdb Guide to Database Design and Definition*.

- 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. These restrictions affect performance with respect to I/O operations.

  The optimizer Index Only Retrieval strategy is disabled if any field in the index has a collating sequence defined, or is a VARCHAR 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 wasdefined 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 data type may be preferable to VARCHAR if the field is involved in index retrieval.

  The following example demonstrates this simple case. The optimizer strategy is displayed when the SET FLAGS statement to 'STRATEGY '.

```
SQL> show table PRODUCTS
Information for table PRODUCTS
Columns for table PRODUCTS:
Column Name                    Data Type        Domain
-----------                    ---------        ------
PRODUCT_ID_V                   VARCHAR(20)
PRODUCT_ID_T                   CHAR(20)
   .
   .
   .
Indexes on table PRODUCTS:
P_INDEX_T                      with column PRODUCT_ID_T
  Duplicates are allowed
  Type is Sorted
  Key suffix compression is DISABLED

 Partition information for index:
  Implicitly mapped to the default storage area
```

```
P_INDEX_V                       with column PRODUCT_ID_V
  Duplicates are allowed
  Type is Sorted
  Key suffix compression is DISABLED

 Partition information for index:
  Implicitly mapped to the default storage area
   .
   .
   .
SQL>
SQL> set flags 'strategy,max_stability';
SQL>
SQL> select product_id_t
cont> from PRODUCTS
cont> where product_id_t starting with 'AAA';
Conjunct        Index only retrieval of relation PRODUCTS
  Index name  P_INDEX_T [1:1]
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 [1:1]
0 rows selected
```

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 at the end 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 (the new range has no data in the table), you can quickly alter indexes that do not contain overflow partitions.

For more information, see the *Oracle Rdb 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.

- 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
```

- The CREATE INDEX statement supplies a default index node size if none is provided for a UNIQUE SORTED index or a SORTED RANKED index. Use the SQL SHOW INDEX or SHOW TABLE statement or the RMU Extract command to display the value of this default node size.

- 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
```

- When a column is defined with a collating sequence, the index key is specially encoded to incorporate the correct ordering (collating) information. This special encoding takes more space than keys encoded for ASCII (the default when no collating sequence is used). Therefore, the encoded string uses more than the customary one byte per character of space within the index.

  For all collating sequences, except Norwegian, the space required is approximately 9 bytes for every 8 characters. So, a CHAR (24) column will require approximately 27 bytes to store. For Norwegian collating sequences, the space required is approximately 10 bytes for every 8 characters.

The space required for encoding the string must be taken into account, when calculating the size of an index key against the limit of 255 bytes. Suppose a column defined with a collating sequence of GERMAN were used in an index. The length of that column is limited to a maximum of 225 characters, since the key will be encoded in 254 bytes.

The following example demonstrates how a 233 character column, defined with a German collating sequence and included in an index, exceeds the index size limit of 255 bytes, even though the column is defined as less than 255 characters in length.

```
SQL> create database
cont>      filename 'TESTDB.RDB'
cont>      Collating sequence GERMAN German;
SQL> create table EMPLOYEE_INFO (
cont>      EMP_NAME CHAR (233));
SQL> create index EMP_NAME_IDX
cont>      on EMPLOYEE_INFO (
cont>      EMP_NAME     asc)
cont>      type is sorted;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-INDTOOBIG, requested index is too big
```

- NOLOGGING is a transient attribute and reverts to LOGGING when the CREATE INDEX statement is committed.
- NOLOGGING has the advantage of reducing the amount of I/O to the recovery-unit journal file (.ruj) and after-image journal files (.aij). However, when the transaction is rolled back, Rdb must use the management records in the journals to undo the Create Index. For UNIFORM format storage areas, this is quite fast because of the **fast logical area delete** mechanism inherent in this area type. On the other hand, MIXED format storage areas require recovery to scan and erase the index data; this additional recovery time should be considered when using this clause.
- When **compression octets** is specified, the character strings can be simple strings, such as '*-!', or if the characters are non-printing, then use the hex string specification, such as X'00A1'.
- The REVERSE attribute is applied to an index to move the I/O for different INSERT processes to different parts of the index and so reduce contention.

  For example, consider a table with a PRIMARY KEY defined on a column with the IDENTITY attribute. Such columns act as AUTOMATIC INSERT AS columns using a sequence and so generate adjacent values. As applications insert into this table, these sequences generated new values which would naturally be inserted in the same part of the SORTED (or SORTED RANKED) index defined on the PRIMARY KEY column. Thus

the leading edge of the index is a focal point (or hot spot) of index updates and often results in contention between competing INSERT users.

- For retrieval via a REVERSE index, only an exact key match is supported, either using the IS NULL or equality (=) operators.
- A REVERSE key index no longer orders the key values by their natural values and therefore there are several restriction on queries using this index.
  * The index cannot be used to return rows ordered by the column value. A true sort operation will be required to implement an ORDER BY, GROUP BY or DISTINCT operation.
  * Range retrieval (BETWEEN, <, >, <=, >= operators) cannot use this type of index.
  * Partial key lookup is not supported. The entire index key value is required so for multi-column indices, all columns must be compared to a value.
  * Column comparisons using STARTING WITH, MATCHING, and LIKE cannot make use of a REVERSE index.
  * MIN and MAX optimizations are not used nor are they possible. The "Max key lookup" and "Min key lookup" optimizations rely on the index being in sorted order by the column being referenced.
  * COUNT and COUNT (DISTINCT) optimizations are not used.
  * The STORE USING clause cannot be used to partition the data since the WITH LIMIT OF clauses would also need to be reversed.

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 truncated 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$SYSTEM (THRESHOLDS ARE (60,75,90))
cont>             WITH LIMIT OF ('00200')
cont>         IN UNIFORM1 (THRESHOLD IS (65))
cont>             WITH LIMIT OF ('00400')
cont>         OTHERWISE IN UNIFORM2
cont>             (THRESHOLD OF (90));
%RDB-W-META_WARN, metadata successfully updated with the reported warning
-RDMS-W-IDXCOLEXIST, an index with this column list already exists
SQL> --
SQL> SHOW INDEX EMP_THRESHOLDS
Indexes on table EMPLOYEES:
EMP_THRESHOLDS                  with column EMPLOYEE_ID
  No Duplicates allowed
  Type is Sorted
  Key suffix compression is DISABLED
  Node size  430
 Store clause:  STORE USING (EMPLOYEE_ID)
        IN RDB$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)
```

Example 10: Using the Index Attributes Clause

```
SQL> CREATE UNIQUE INDEX JOB_JOB_CODE
cont> ON JOBS (
cont> JOB_CODE
cont> ASC)
cont> TYPE IS SORTED
cont> THRESHOLDS ARE (75,83,90)
cont> ENABLE COMPRESSION
cont> NOLOGGING
cont> COMMENT IS 'Used for translation of job codes';
%RDB-W-META_WARN, metadata successfully updated with the reported warning
-RDMS-W-DATACMIT, unjournaled changes made; database may not be recoverable
SQL> -- SQL returned this message because the NOLOGGING attribute
SQL> -- was set.
```

Example 11: Creating an Index and Displaying the Default Node Size

```
SQL> -- Create a simple table upon which to define
SQL> -- some indexes
SQL>
SQL> CREATE TABLE TEST_INDEX_TABLE
cont>    (A CHAR(70),
cont>     B INTEGER);
SQL>
SQL> -- Default value is 430 bytes
SQL>
SQL> CREATE UNIQUE INDEX TEST_INDEX_DEF
cont>    ON TEST_INDEX_TABLE (A, B)
cont>    TYPE IS SORTED
cont>    USAGE UPDATE;
SQL>
SQL> SHOW TABLE (INDEX) TEST_INDEX_TABLE
Information for table TEST_INDEX_TABLE
TEST_INDEX_DEF                  with column A
                                and column B
  No Duplicates allowed
  Type is Sorted
  Compression is DISABLED
  Node size  430
  Percent fill  70
```

Example 12: Naming Partitions

```
SQL> -- Alter mf_personnel database to add three slots
SQL> -- for storage areas and then add three storage areas.
SQL> ALTER DATABASE FILENAME MF_PERSONNEL
cont> RESERVE 3 STORAGE AREAS;
%RDMS-W-DOFULLBCK, full database backup should be done to ensure future recovery
SQL> ALTER DATABASE FILENAME MF_PERSONNEL
cont> ADD STORAGE AREA WAGE_LOW;
SQL> ALTER DATABASE FILENAME MF_PERSONNEL
cont> ADD STORAGE AREA WAGE_MID;
SQL> ALTER DATABASE FILENAME MF_PERSONNEL
cont> ADD STORAGE AREA WAGE_HIGH;
SQL> ATTACH 'FILENAME MF_PERSONNEL.RDB';
SQL> -- Create an index on the JOBS table and name the partitions
SQL> CREATE INDEX WAGE_CLASS_IDX ON JOBS (WAGE_CLASS)
cont> TYPE IS SORTED
cont> STORE USING (WAGE_CLASS)
cont> IN WAGE_LOW (PARTITION WAGE_LOW) WITH LIMIT OF ('1')
cont> IN WAGE_MID (PARTITION WAGE_MID) WITH LIMIT OF ('3')
cont> OTHERWISE IN WAGE_HIGH (PARTITION WAGE_HIGH);
```

Example 13: Creating a Large Index Partitioned Across Many Storage Areas

First, create the database definition:

```
SQL> CREATE INDEX ... MAINTENANCE IS ENABLED DEFERRED ...;
```

Next submit batch jobs to build each partition in parallel. For example, each batch job would execute a script similar to the following:

```
ATTACH 'filename testdatabase';
SET FLAGS 'index_stats';
ALTER INDEX TRANSACTIONS_INDEX BUILD PARTITION PART_1;
COMMIT;
```

Finally, after the batch jobs have completed, the database administrator must make the index active for query usage by changing the maintenance mode to ENABLED IMMEDIATE. A BUILD ALL PARTITIONS clause could be added in case any step failed (possibly due to resource limitations or a failed node).

```
SQL> SET FLAGS 'index_stats';
SQL> SET TRANSLATION READ WRITE RESERVING...FOR EXCLUSIVE WRITES;
SQL> ALTER INDEX ... BUILD ALL PARTITIONS;
SQL> ALTER INDEX ... MAINTENANCE IS ENABLED IMMEDIATE;
SQL> COMMIT;
```

This scheme has several advantages over issuing a CREATE INDEX statement directly:

- The build actions can be run in parallel, which allows better resource usage (read and sort fewer rows), and reduced execution time for the index creation.
- The partitions being processed are relatively small when compared to the full index and, therefore, smaller quantities of data will be processed. This will result in smaller .ruj files and less AIJ file space for these transactions.
- Each build partition runs in a separate transaction, can easily be repeated if a step fails, and does not require repeating the entire CREATE INDEX statement.
- If any steps have failed, they will also be repeated by the BUILD ALL PARTITIONS clause included in the script.

Example 14: Showing the use of **compression octets** clause

Here the database administrator noticed that -1 was used as a flag in the table, and many rows use the DEFAULT value. It makes sense to include x'FF' as a compression character.

```
SQL> create table PEOPLE
cont>     (name    char (100) default ' '
...
cont>     ,use_indicator      bigint default -1
cont>     )
cont> ;
SQL>
SQL> create index PEOPLE_NDX3
cont>     on PEOPLE (use_indicator)
cont>     enable compression
cont>         (minimum run length 2,
cont>          compression octets (X'00', X'FF'))
cont>     store in PEOPLE_NDX3_AREA
cont> ;
SQL>
SQL> insert into PEOPLE
cont>     default values;
1 row inserted
SQL>
```

Example 15

The following example shows the definition of tables and indices with REVERSE key encoding. The PRIMARY KEY and FOREIGN KEY definitions use these indices to perform validation during INSERT.

```
SQL> create table ACTION_CODES
cont>     (actioncode integer primary key
cont>     );
SQL>
SQL> begin
cont> -- load the ACTION_CODES table
cont> declare :l integer;
cont> for :l in 1 to 50 do
cont>     insert into ACTION_CODES values (:l);
cont> end for;
cont> end;
SQL>
SQL> create unique index ACTION_CODES_PK
cont>     on ACTION_CODES (actioncode)
cont>     type is SORTED RANKED
cont>     reverse;
SQL>
SQL> create table LOG_TABLE
cont>     (ident integer identity primary key
cont>     ,tstamp timestamp(2)
cont>     ,actioncode integer references ACTION_CODES (actioncode)
cont>     ,description varchar(300)
cont>     );
SQL>
SQL> create unique index LOG_TABLE_PK
cont>     on LOG_TABLE (ident)
cont>     type is SORTED RANKED
cont>     reverse;
SQL>
SQL> -- real values
SQL> insert into LOG_TABLE
cont>     values (timestamp'2014-08-18 14:31:07.46',
cont>             42, 'correct spelling of address field data');
1 row inserted
SQL>
SQL> -- detect bad foreign value
SQL> insert into LOG_TABLE
cont>     values (timestamp'2014-08-18 14:31:07.46', 142, 'data');
%RDB-E-INTEG_FAIL, violation of constraint LOG_TABLE_FOREIGN1 caused operation
to fail
-RDB-F-ON_DB, on database RDB$DEFAULT_CONNECTION
SQL>
```

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 written in SQL 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.

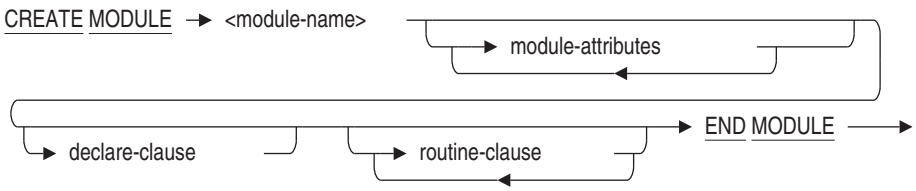
In general, SQL operates on modules, not stored routines. However, there are a few exceptions: DROP FUNCTION, DROP PROCEDURE, RENAME, SHOW FUNCTION, SHOW PROCEDURE, and CALL. The SHOW FUNCTION statement displays information about functions. The SHOW PROCEDURE statement displays individual procedures in a stored module. The CALL statement can invoke only a single stored procedure.

Environment

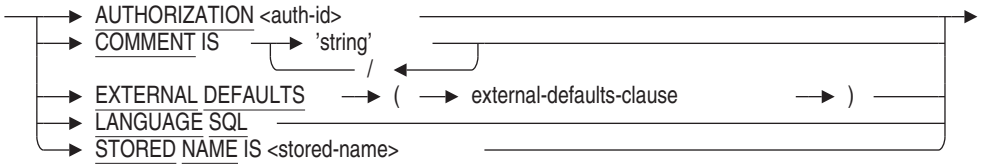
You can use the CREATE MODULE statement in a simple statement procedure:

- 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

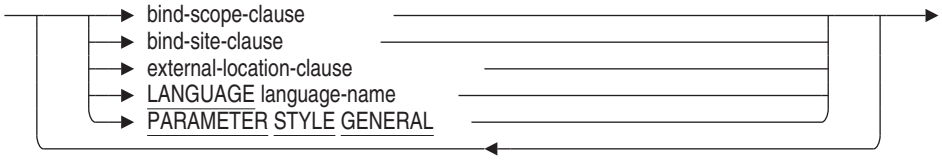
Format



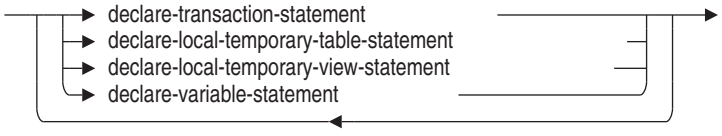
module-attributes =



external-defaults-clause =

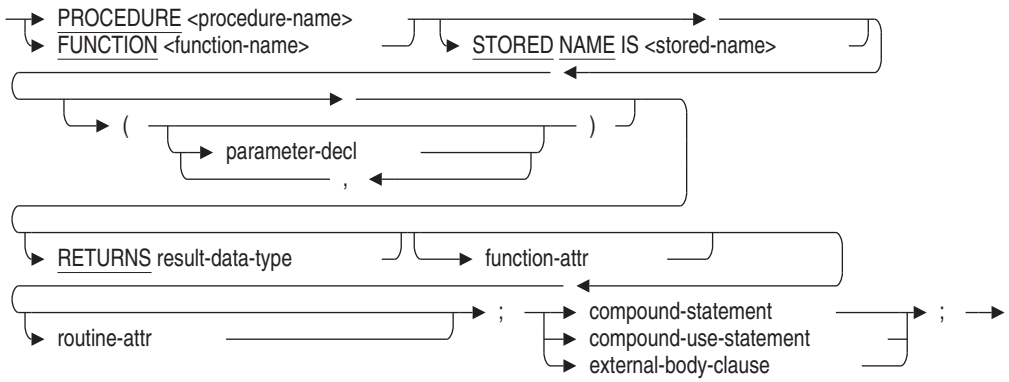


declare-clause =

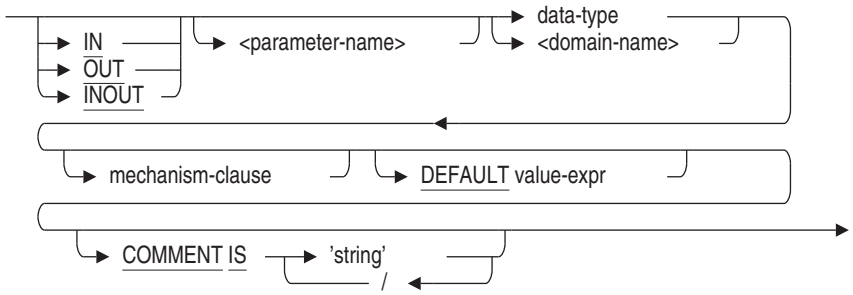


CREATE MODULE Statement

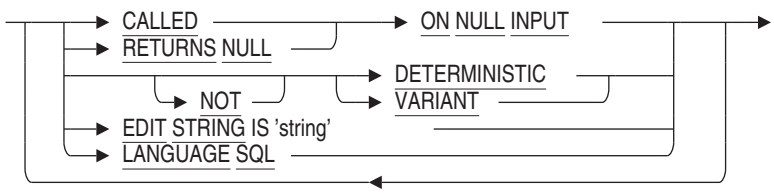
routine-clause =



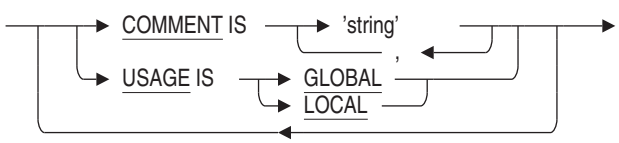
parameter-decl =



function-attr =



routine-attr =



Arguments

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 2.2.2 for information about using authorization identifiers.

CALLED ON NULL INPUT
This is the default. The SQL function should be prepared to handle input arguments (IN and INOUT) that are NULL.

COMMENT IS 'string'
Adds a comment about the module, routine, and parameter. 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 (/).

compound-statement
Allows you to include more than one SQL statement in a stored routine. See the Compound Statement for more information.

compound-use-statement
Allows you to include one SQL statement in a stored routine. See the Compound Statement for more information.

If you are defining a stored function, the simple statement must be the RETURNS clause.

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.

declare-local-temporary-table-statement
Declares a local temporary table for the module. See the DECLARE LOCAL TEMPORARY TABLE Statement for more information.

declare-local-temporary-view-statement
Declares a local temporary view for the module. See the DECLARE LOCAL TEMPORARY VIEW Statement for more information.

declare-transaction-statement
Declares a transaction for the module. Only one declare-transaction-statement is permitted for each module. If omitted, an implicit DECLARE TRANSACTION READ WRITE is used. See the DECLARE TRANSACTION Statement for more information.

declare-variable-statement
Declares a global variable for the module. See the DECLARE Variable Statement for more information.

DEFAULT value-expr
Specifies the default value of a parameter for a function or procedure defined with mode IN. If you omit this parameter or if the Call statement argument list or function invocation specifies the DEFAULT keyword, then the value-expr specified with this clause is used. The parameter uses NULL as the default if you do not specify a value expression explicitly.

DETERMINISTIC
NOT DETERMINISTIC
The clause controls the evaluation of an external function in the scope of a query:

- NOT DETERMINISTIC

  Specifying the NOT DETERMINISTIC 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 DETERMINISTIC clause.

- DETERMINISTIC

  Specifying the DETERMINISTIC 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 DETERMINISTIC 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 DETERMINISTIC, the function F1() may be evaluated just once depending on the optimizer. If you define the F1 function as NOT DETERMINISTIC, the function F1() is evaluated twice.

  DETERMINISTIC is the default.

The DETERMINISTIC or NOT DETERMINISTIC clause is not allowed on procedure definitions.

domain-name
The name of a domain created in a CREATE DOMAIN statement. For more information about domains, see the CREATE DOMAIN Statement.

DEFAULT value-expr
Specifies the default value of a parameter for a function or procedure defined with mode IN. If you omit this parameter or if the Call statement argument list or function invocation specifies the DEFAULT keyword, then the value-expr specified with this clause is used. The parameter uses NULL as the default if you do not specify a value expression explicitly.

EDIT STRING is 'string'
Allows the value returned from the function invocation to be implicitly formatted using the EDIT STRING associated with the function. This is similar to defining an EDIT STRING on a column or domain. The EDIT STRING clause is only used by queries in Interactive SQL.

external-body-clause
Identifies key characteristics of the routine: its name, where the executable image is located, the language in which the routine is coded, and so forth. Please refer to the CREATE ROUTINE Statement section for further details.

external-defaults-clause
The EXTERNAL DEFAULTS clause allows the user to specify external routine attributes just once in the module header so that duplicated values can be omitted for procedure and function definitions (attributes such as LOCATION, LANGUAGE, PARAMETER STYLE, BIND ... SITE and BIND SCOPE clauses). See the Create Routine statement for more information on these clauses.

IN
OUT
INOUT parameter-name
Specifies the parameter modes used in a 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 functions.

Each parameter name must be unique within the routine.

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.

module-name
A user-supplied name that you assign to a module.

See Section 2.2 for more information on user-supplied names.

parameter-decl
Specifies the parameters and parameter modes used in a stored or external routine.

PROCEDURE procedure-name
FUNCTION function-name
A user-supplied name that you give to a stored or external routine in a module. The name you specify for a stored routine must be unique within the database definition.

RETURNS NULL ON NULL INPUT
Indicates that Oracle Rdb will avoid calling the SQL function and return NULL as the function result if a NULL value is passed to any parameter of the function.

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 function.

routine-clause
The definition of a stored function or stored procedure created in a module.

STORED NAME IS stored-name
Specifies a name that Oracle Rdb uses to access a module procedure or function created in a multischema database.

USAGE IS
Specifies how the function or procedure can be called:

- USAGE IS GLOBAL indicates that the function or procedure can be called outside the current module. This is the default.

- USAGE IS LOCAL specifies that the routine is restricted to references within the module.

VARIANT
NOT VARIANT
These clauses are synonyms for the DETERMINISTIC and NOT DETERMINISTIC clauses. The DETERMINISTIC clause indicates that the same inputs to the function will generate the same output. It is the same as the NOT VARIANT clause. The NOT DETERMINISTIC clause indicates that the output of the function does not depend on the inputs. It is the same as the VARIANT clause.

This clause is deprecated. Use DETERMINISTIC instead.

Usage Notes

- The full syntax diagrams for external routines are described in the CREATE ROUTINE statement.
- 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.
- When the AUTHORIZATION clause is used, the definer of the module is granting their own privileges to the specified username so that tables, columns, sequences, procedures and functions are accessed as though accessed by the definer.

The AUTHORIZATION is expected to be the session user, or an OpenVMS rights identifier granted to that user (when SECURITY CHECKING IS EXTERNAL) or a role defined in the database and granted to that user (when SECURITY CHECKING IS INTERNAL). If the session is run with one of the following OpenVMS privileges, then any user or rights identifier can be referenced: SYSPRV, BYPASS or IMPERSONATE.

- 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.
- If the parameter mode is omitted, it defaults to IN for external routines and stored functions. For stored procedures it is determined by usage. Use SHOW PROCEDURE and SHOW FUNCTION to see these implicit settings.
- 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 6–4 shows which statements can cause SQL to invalidate stored routines.

Table 6–4 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
	ALTER TABLE ALTER COLUMN AFTER COLUMN	No	Yes	LS
	ALTER TABLE ALTER COLUMN BEFORE COLUMN	No	Yes	LS
Constraint	ALTER TABLE DROP CONSTRAINT	Yes	No	SR
	ALTER TABLE ADD CONSTRAINT	No	No	SR or DE
	RENAME CONSTRAINT	No	No[6]	DE

[1]Dependency types: **DE** (default evaluating), **DR** (default reserving), **LS** (language semantics), **SR** (stored routine), **SM** (stored module).

[6]If the old constraint name is referenced in a DECLARE TRANSACTION ... EVALUATING clause and later this declasred transaction is used when invoking a routine an error will be raised.

(continued on next page)

Table 6–4 (Cont.) 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]
Domain	ALTER DOMAIN data type (in parameter list)	Yes	No	SR[2]
	ALTER DOMAIN data type (in procedure block)	No	No	SR[3]
	DROP DOMAIN	Yes	No	SR or SM
	RENAME DOMAIN	No	No[4]	SR
Function	ALTER FUNCTION	No	No	SR
	DROP FUNCTION RESTRICT	Yes	No	SR
	DROP FUNCTION CASCADE	No	Yes	SR
	RENAME FUNCTION	No	No[4]	SR
Module	ALTER MODULE DROP FUNCTION RESTRICT	Yes	No	SR
	ALTER MODULE DROP FUNCTION CASCADE	No	Yes	SR
	ALTER MODULE DROP PROCEDURE RESTRICT	Yes	No	SR
	ALTER MODULE DROP PROCEDURE CASCADE	No	Yes	SR
	DROP MODULE RESTRICT	Yes	No	SR
	DROP MODULE CASCADE	No	Yes	SR
	RENAME MODULE	No	No[4]	SR
Procedure	ALTER PROCEDURE	No	No	SR
	DROP PROCEDURE RESTRICT	Yes	No	SR
	DROP PROCEDURE CASCADE	No	Yes	SR
	RENAME PROCEDURE	No	No[4]	SR

[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.

[4]RENAME adds a synonym for the old name, only if this synonym is dropped is the routine invalidated.

(continued on next page)

Table 6–4 (Cont.) 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]
Sequence	ALTER SEQUENCE	No	No	SR
	DROP SEQUENCE RESTRICT	Yes	No	SR
	DROP SEQUENCE CASCADE	No	Yes	SR
	RENAME SEQUENCE	No	No[4]	SR
Synonym	ALTER SYNONYM	No	No[5]	SR
	DROP SYNONYM RESTRICT	Yes	No	SR
	DROP SYNONYM CASCADE	No	Yes	SR
Table	ALTER TABLE	No	No	SR
	DROP TABLE RESTRICT	Yes	No	SR, LS, DR, or SM
	DROP TABLE CASCADE	No	Yes	SR or LS
	RENAME TABLE	No	No[4]	SR
View	DROP VIEW RESTRICT	Yes	No	SR, LS, or DR
	DROP VIEW CASCADE	No	Yes	SR or LS
	RENAME VIEW	No	No[4]	SR

[1]Dependency types: **DE** (default evaluating), **DR** (default reserving), **LS** (language semantics), **SR** (stored routine), **SM** (stored module).

[4]RENAME adds a synonym for the old name, only if this synonym is dropped is the routine invalidated.

[5]The altered synonym should reference an object (table, view, function or procedure) with the same column or parameter list, otherwise an error will occur at run-time.

- The maximum length for each string literal in a comment is 1,024 characters.
- You cannot specify the COMMIT, ROLLBACK, START TRANSACTION, or SET TRANSACTION statements within a stored function.
- You can invoke a function anywhere a value expression is allowed.
- 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. See also the DECLARE STATEMENT Statement and DECLARE Routine Statement.
- The TRACE statement 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.
- In general, SQL operates on modules, not on the contained routines, with the following 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.
- The mechanism-clause is not permitted for SQL stored functions or procedures.
- The following usage notes provide information about global variables:
  - Global variables can be referenced from any routine within the created module (just like local variables).
  - If a local variable and a global variable have the same name, then the local variable takes precedence over the global variable within the scope of the function or procedure in which the local variable is declared. For example:

```
SQL> SET FLAGS 'TRACE';
SQL> CREATE MODULE SAMPLE
cont>    DECLARE :IMAX INTEGER DEFAULT 100
cont>    PROCEDURE TRACE_MAX;
cont>    BEGIN
cont>    DECLARE :IMAX INTEGER DEFAULT 0;
cont>    TRACE :IMAX;
cont>    END;
cont> END MODULE;
SQL> CALL TRACE_MAX();
~Xt: 0
```

  - All DEFAULT value clauses are evaluated when the first function or procedure is called for a module.
  - Data persists until a module is unloaded at DISCONNECT time. Therefore, routines in a module can exchange data using global variables.
  - Neither a COMMIT nor a ROLLBACK statement affects the state of the variables. That is, changes to variables are not rolled back.

- When CURRENT_USER or CURRENT_UID are used as a default for a stored function or stored procedure, they are evaluated as though passed by the caller, and are not evaluated within the routine being created. This is significant only if the module is a definers rights module (that is, has an AUTHORIZATION clause).

  For example, the following example returns the invoker as the current user and not the AUTHORIZATION of the module as might be expected:

```
SQL> create module DEF
cont>     authorization FEENAN
cont> procedure DEF1
cont>     (in :a char(31)
cont>      default current_user);
cont> trace :a;
cont> end module;
SQL>
SQL> set flags 'trace';
SQL> begin
cont> call DEF1 ();
cont> end;
~Xt: SMITH
```

- If the default expression uses SQL functions to access tables, then those tables should be explicitly locked within the function or nested called procedures using the LOCK TABLE statement within the function.
- External and stored routines can be included in a single module. This provides the following benefits:
  - This allows simplified DROP, GRANT and REVOKE operations which will operate on multiple routines in a single statement. For instance, a DROP MODULE can be used to remove external and stored routines in a single command. GRANT and REVOKE can be applied to a larger set of routines, rather than requiring individual statements for each external routine.
  - Related routines, whether external or stored, can be grouped together thus providing simplified database maintenance.
  - External routines within the same module now share the same database environment. This allows, for instance, one external routine to OPEN a cursor, another to FETCH rows and another to CLOSE the cursor.

    In contrast when an external routine is created using the CREATE FUNCTION or CREATE PROCEDURE syntax only that routine uses the database environment.

- The name of the stored module need not be the same as that used for the SQL module language module, or SQL pre-compiled module which implements the external functionality. However, keeping identical or similar names may assist future maintenance of the application.
- The shared module database environment is only significant for external routines which execute SQL statements.
- If an external routine attaches to a database, it will be implicitly disconnected when the invoking session is terminated.

  However, Oracle recommends that the current transaction, open cursors and session started for the external function be terminated before using DISCONNECT. This can be done explicitly by calling an external routine which terminates the transaction and disconnects in the same context as the invoking routine, or it can be done implicitly when using a NOTIFY routine.
- If declare-transaction-statement is omitted from the CREATE MODULE definition then SQL applies a default of DECLARE TRANSACTION READ WRITE
- The module attributes can appear in any order following the name of the module and before the DECLARE statements.

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> COMMIT;
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 SQL module language

The following code fragment 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 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;
```

Example 7: Declaring a Global Variable to Exchange Information Between Two Routines

```
SQL> CREATE MODULE sample
cont>  LANGUAGE SQL
cont>  DECLARE :iter_count INTEGER
cont>  PROCEDURE set_iter (IN :val INTEGER)
cont>  COMMENT IS 'Validate the iteration count and assign'
cont>  /          'to a global variable.';
cont>  BEGIN
cont>  IF (:val IS NULL) OR (:val < 1) THEN
cont>     SIGNAL 'XXXXX'; --illegal value
cont>  ELSE
cont>     SET :iter_count  =:val;
cont>     TRACE 'Iteration count set to ', :val;
cont>  END IF;
cont>  END;
cont> FUNCTION GET_ITER ()
cont> RETURNS INTEGER
cont> COMMENT IS 'Trace the value used and then return the'
cont> / 'value from the global variable.';
cont> BEGIN
cont> TRACE 'Using iteration count ', :iter_count;
cont> RETURN :iter_count;
cont> END;
cont> END MODULE;
```

Example 8: Using a cursor implemented by external routines

This example uses multiple external routines to manage a table cursor in the external routine database environment. This management includes the OPEN, FETCH and CLOSE of a single cursor.

Several domains are defined so that parameter data types can be consistently defined in the database that contain the application and also the database upon which the cursor is open.

```
create domain SQLSTATE_T char(5);
create domain STATUS_CODE char(1);
create domain STATUS_NAME char(8);
create domain STATUS_TYPE char(14);
```

The external function interface is contained within a single CREATE MODULE statement. This module also contains the application in a single stored SQL procedure.

```
create module EX
    language SQL

    -- These procedure define the interface to the external
    -- routines that implement the transaction and cursor operations
    --
    procedure EX_START_READ_TXN
        (inout :ss sqlstate_t);
        external location 'TEST$SCRATCH:EX.EXE'
        language general
        general parameter style
        comment is 'start a READ ONLY transaction';

    procedure EX_COMMIT
        (inout :ss sqlstate_t);
        external location 'TEST$SCRATCH:EX.EXE'
        language general
        general parameter style;

    procedure EX_OPEN_CURSOR
        (inout :ss sqlstate_t);
        external location 'TEST$SCRATCH:EX.EXE'
        language general
        general parameter style
        comment is 'find all rows in WORK_STATUS order by STATUS_CODE';

    procedure EX_CLOSE_CURSOR
        (inout :ss sqlstate_t);
        external location 'TEST$SCRATCH:EX.EXE'
        language general
        general parameter style;

    procedure EX_FETCH_CURSOR
        (inout :ss sqlstate_t,
         out :s_code STATUS_CODE, out :s_code_ind integer,
         out :s_name STATUS_NAME, out :s_name_ind integer,
         out :s_type STATUS_TYPE, out :s_type_ind integer);
        external location 'TEST$SCRATCH:EX.EXE'
        language general
        general parameter style;

    -- This SQL procedures implements a simple application
    --
    procedure WORK_STATUS
        comment is  'Use an external cursor to fetch all rows in the'
        /           'WORK_STATUS table';
        begin
        declare :s_code STATUS_CODE;
        declare :s_name STATUS_NAME;
        declare :s_type STATUS_TYPE;
        declare :s_code_ind, :s_name_ind, :s_type_ind integer;
        declare :ss sqlstate_t;
```

```
-- start a read-only transaction on the PERSONNEL database
call EX_START_READ_TXN (:ss);
if :ss ^= '00000' then
    SIGNAL :ss;
end if;

-- open the cursor on the work-status table
call EX_OPEN_CURSOR (:ss);
if :ss ^= '00000' then
    SIGNAL :ss;
end if;

-- now loop and fetch all the rows
FETCH_LOOP:
loop
    call EX_FETCH_CURSOR (:ss,
                          :s_code, :s_code_ind,
                          :s_name, :s_name_ind,
                          :s_type, :s_type_ind);
    case :ss
        when '02000' then
            -- no more rows to fetch
            leave FETCH_LOOP;

        when '00000' then
            begin
            -- we have successfully fetched a row, so display it
            trace 'Status Code: ', case when :s_code_ind < 0
                                        then 'NULL'
                                        else :s_code
                                   end;
            trace 'Status Name: ', case when :s_name_ind < 0
                                        then 'NULL'
                                        else :s_name
                                   end;
            trace 'Status Type: ', case when :s_type_ind < 0
                                        then 'NULL'
                                        else :s_type
                                   end;
            trace '***';
            end;
        else
            -- signal will implicitly leave the stored procedure
            SIGNAL :ss;
    end case;
end loop;

-- close the cursor
call EX_CLOSE_CURSOR (:ss);
if :ss ^= '00000' then
    SIGNAL :ss;
end if;
```

```
        -- commit the transaction
        call EX_COMMIT (:ss);
        if :ss ^= '00000' then
            SIGNAL :ss;
        end if;
        end;

end module;
```

The external procedures for this this example are written using the SQL module language. However, any language with embedded SQL, such as C, could have been used.

```
module EX
language GENERAL
parameter colons
-- EX: Sample application
-- Process the WORK_STATUS table using a table cursor
--
declare alias filename 'PERSONNEL'

declare c cursor for
    select status_code, status_name, status_type
     from WORK_STATUS
     order by status_code

procedure EX_START_READ_TXN
    (sqlstate);
    begin
    -- abort any stray transactions
    rollback;
    -- start a READ ONLY transaction
    set transaction read only;
    end;

procedure EX_COMMIT
    (sqlstate);
    commit work;

procedure EX_ROLLBACK
    (sqlstate);
    rollback work;

procedure EX_OPEN_CURSOR
    (sqlstate);
    open c;

procedure EX_CLOSE_CURSOR
    (sqlstate);
    close c;
```

```
procedure EX_FETCH_CURSOR
    (sqlstate,
     :s_code            STATUS_CODE,
     :s_code_ind        integer,
     :s_name            STATUS_NAME,
     :s_name_ind        integer,
     :s_type            STATUS_TYPE,
     :s_type_ind        integer);
    fetch c
     into :s_code indicator :s_code_ind,
          :s_name indicator :s_name_ind,
          :s_type indicator :s_type_ind;

procedure EX_DISCONNECT
    (sqlstate);
    disconnect default;
```

When run the application calls the external procedures to open the cursor and fetch the rows and display them using the TRACE statement.

```
SQL> set flags 'trace';
SQL>
SQL> call WORK_STATUS ();
~Xt: Status Code: 0
~Xt: Status Name: INACTIVE
~Xt: Status Type: RECORD EXPIRED
~Xt: ***
~Xt: Status Code: 1
~Xt: Status Name: ACTIVE
~Xt: Status Type: FULL TIME
~Xt: ***
~Xt: Status Code: 2
~Xt: Status Name: ACTIVE
~Xt: Status Type: PART TIME
~Xt: ***
SQL>
```

Oracle recommends that the cursors be closed, and the external routines database environment be disconnected before the calling session is disconnected. This can be achieved by using NOTIFY routines.

For example, the external procedure that starts the transaction could be modified as shown below to declare a NOTIFY routine (EX_RUNDOWN) that when called would close the cursors, rollback the transaction and disconnect from the database.

```
    procedure EX_START_READ_TXN
        (inout :ss sqlstate_t);
        external location 'TEST$SCRATCH:EX.EXE'
        language general
        general parameter style
        notify EX_RUNDOWN on BIND
        comment is 'start a READ ONLY transaction';
```

The BIND notification ensures that EX_RUNDOWN will be called during the DISCONNECT of the caller and allow the transaction to be rolled back and the session disconnected. ROLLBACK or COMMIT will implicitly close any open cursors, unless the cursor were defined as WITH HOLD. In this case it is important to also close that cursor. Code similar to the following (in C) could implement this rundown routine.

```
#include <string.h>
#include <stdio.h>
#define RDB$K_RTX_NOTIFY_ACTV_END 2
#define SQLSTATE_LEN 5
void sql_signal ();
void EX_CLOSE_CURSOR (char sqlstate [SQLSTATE_LEN]);
void EX_DISCONNECT (char sqlstate [SQLSTATE_LEN]);
void EX_ROLLBACK (char sqlstate [SQLSTATE_LEN]);

extern void EX_RUNDOWN
    (int *func_code,
     int *u1,                   /* U1, U2, U3 are currently unused */
     int *u2,                   /* and are reserved for future use */
     int *u3)
{
char sqlstate [SQLSTATE_LEN];

    if (*func_code == RDB$K_RTX_NOTIFY_ACTV_END)
    {
        /* we are running down this external routine, so
         * close the cursor
         */
        EX_CLOSE_CURSOR (sqlstate);
        if (memcmp ("00000", sqlstate, SQLSTATE_LEN) != 0
            && memcmp ("24000", sqlstate, SQLSTATE_LEN) != 0)
            /* we expect success or maybe 24000 (bad cursor state)
             */
            sql_signal ();

        /* rollback the transaction
         */
        EX_ROLLBACK (sqlstate);
        if (memcmp ("00000", sqlstate, SQLSTATE_LEN) != 0
            && memcmp ("25000", sqlstate, SQLSTATE_LEN) != 0)
            /* we expect success or maybe 25000 (bad transaction state)
             */
            sql_signal ();
```

```
        /* disconnect from the database
         */
        EX_DISCONNECT (sqlstate);
        if (memcmp ("00000", sqlstate, SQLSTATE_LEN) != 0)
            /* we expect success or maybe 25000 (bad transaction state)
             */
            sql_signal ();
    }
}
```

The application can be compiled and built using this fragment of DCL code:

```
$       create ex.opt
symbol_vector = (EX_START_READ_TXN = procedure)
symbol_vector = (EX_COMMIT = procedure)
symbol_vector = (EX_ROLLBACK = procedure)
symbol_vector = (EX_OPEN_CURSOR = procedure)
symbol_vector = (EX_CLOSE_CURSOR = procedure)
symbol_vector = (EX_FETCH_CURSOR = procedure)
symbol_vector = (EX_DISCONNECT = procedure)
symbol_vector = (EX_RUNDOWN = procedure)
psect_attr = RDB$MESSAGE_VECTOR,noshr
psect_attr = RDB$DBHANDLE,noshr
psect_attr = RDB$TRANSACTION_HANDLE,noshr
sql$user/library
$
$       cc EX_RUNDOWN
$       sql$mod EX
$       link/share EX,EX_RUNDOWN,EX/option
```

Example 9: Using the EDIT STRING clause to add formatting to what the SQL function returns

```
SQL> create module SAMPLE
cont>     function SHOW_EMP_ID
cont>         (in :last_name varchar(30)
cont>         ,in : birthday date)
cont>     returns integer
cont>     edit string '9999"-"9999"-"99?"VOID"'
cont>     ;
cont>     return
cont>         (select cast(employee_id as integer) * 100
cont>          from EMPLOYEES
cont>          where birthday = :birthday
cont>            and last_name = :last_name);
cont> end module;
SQL>
SQL> select SHOW_EMP_ID (last_name, birthday), last_name, first_name
cont>  from EMPLOYEES
cont>  where employee_id < '00170';
               LAST_NAME        FIRST_NAME
 0000-0164-00   Toliver          Alvin
 0000-0165-00   Smith            Terry
 0000-0166-00   Dietrich         Rick
 0000-0167-00   Kilpatrick       Janet
 0000-0168-00   Nash             Norman
 0000-0169-00   Gray             Susan
6 rows selected
SQL>
SQL> -- demonstrate the output when a NULL result is returned
SQL> select SHOW_EMP_ID ('Unknown', current_date)
cont>  from EMPLOYEES
cont>  fetch first row only;

 VOID
1 row selected
SQL>
```

CREATE OUTLINE Statement

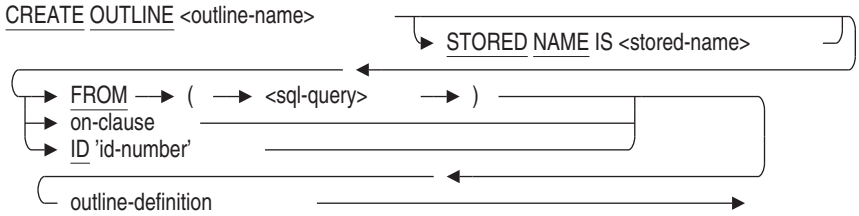
Creates a new query outline and stores this outline in the database.

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.

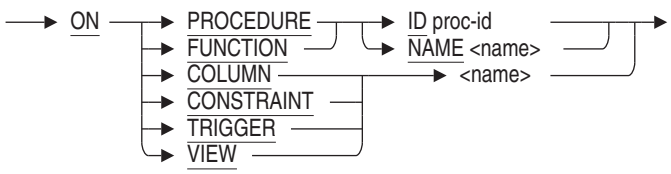
Environment

You can use the CREATE OUTLINE statement only in interactive SQL.

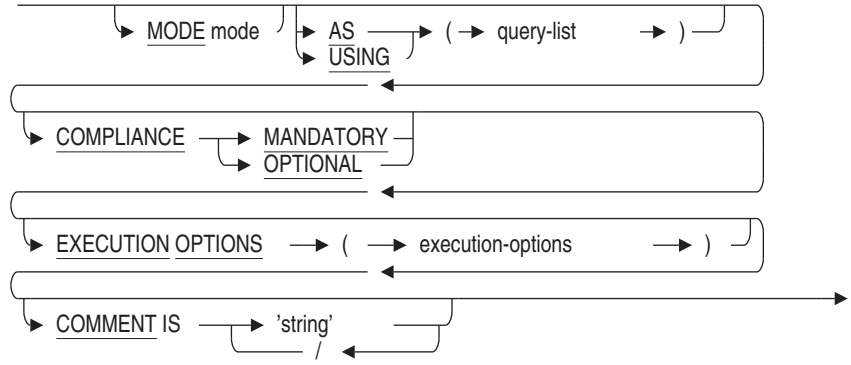
Format



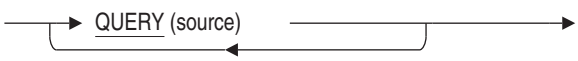
on-clause =



outline-definition =



query-list =



source =

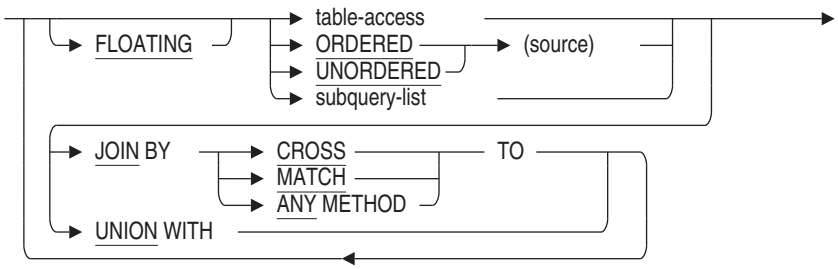
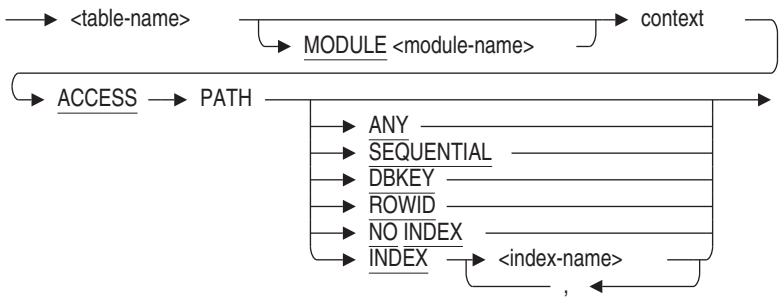
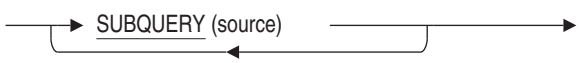


table-access =



subquery-list =



execution-options =

Arguments

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.

Path	Meaning
SEQUENTIAL	Indicates that sequential access should be used.
DBKEY	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]ROWID is a synonym for DBKEY.

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.

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.

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 (/).

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.

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.

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).

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.

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.

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.

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.

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 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 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.

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.

ON COLUMN name
Generates an outline for the specified columns DEFAULT, COMPUTED BY or AUTOMATIC expression. This is a partial outline which will be used when the column is referenced. If a column has both an AUTOMATIC UPDATE AS clause and a DEFAULT expression, then only one outline is created for the AUTOMATIC clause.

ON CONSTRAINT name
Generates an outline definition for the specified constraint.

ON FUNCTION ID proc-id
ON FUNCTION NAME name
Generates an outline definition for the specified stored function.

ON PROCEDURE ID proc-id
ON PROCEDURE NAME name
Generates an outline definition for the specified stored procedure.

ON TRIGGER name
Generates an outline definition for the specified trigger.

ON VIEW name
Generates an outline definition for the specified view.

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.

outline-name
The name of the new query outline. The name has a maximum length of 31 characters.

QUERY
Specifies that the data sources within the parentheses belong to a separate query.

SUBQUERY
Specifies that the data sources within the parentheses belong to a separate subquery.

table-name
Specifies the name of a database table.

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.

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.

USING (query-list)
Specifies the outline to be used for compilation of the contents of the FROM and ON clauses.

You cannot use this clause with the ID id-number clause.

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 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.
- The order of the queries in an outline matches the order of optimization, not the order of execution. The query outline generated by Oracle Rdb appears with comments after the QUERY keyword in the outline to make reading easier. See the Examples section.
- The query outline generated by Oracle Rdb may not have a query corresponding to each statement within the procedure.
- Not all statements require the query optimizer. For example, TRACE and SET statements that do not reference tables do not require the optimizer.
- Subqueries in IF and CASE statements may be lifted into a previous statement by the optimizer to reduce the overhead associated with that query.

- Subqueries within an INSERT or REPLACE statement are executed as though SET statements were performed prior to the INSERT or REPLACE operation.
- INSERT and REPLACE statements are not subject to query optimization.
- During compilation of a constraint or trigger, Oracle Rdb will search for a query outline with the same name as the trigger or constraint being compiled. If a match is found, that outline will be used during the query compilation of the trigger or constraint. If no outline is found matching the name of the object, Oracle Rdb will then try to locate an appropriate outline using the BLR ID of the query. For example:

```
   .
   .
   .
SQL> CREATE TABLE TAB1
cont>     (a1 int CONSTRAINT TAB1NOTNULL NOT NULL DEFERRABLE
cont>     ,a2 char(10)
cont>     ,a3 char(10) );
SQL>
SQL> CREATE INDEX NDX1
cont>     ON TAB1 (a1);
SQL>
SQL> CREATE OUTLINE TAB1NOTNULL
cont>     id '8755644BCB040948E28A76B6D77CC2D3'
cont>     MODE 0
cont>     AS (
cont>       QUERY (
cont>         SUBQUERY (
cont>           TAB1 0 ACCESS PATH SEQUENTIAL
cont>         )
cont>       )
cont>     )
cont>     COMPLIANCE OPTIONAL ;
SQL>
SQL> CREATE TRIGGER TAB1TRIG
cont>     BEFORE INSERT ON TAB1
cont>     (UPDATE TAB1 SET a3= 'bbbb' WHERE a2 = 'aaaa')
cont>     FOR EACH ROW;
SQL>
SQL> CREATE OUTLINE TAB1TRIG
cont>     on trigger TAB1TRIG
cont>     MODE 0
cont>     AS (
cont>       QUERY (
cont>         SUBQUERY (
cont>           TAB1 0 ACCESS INDEX NDX1
cont>         )
cont>       )
cont>     )
```

```
cont>     COMPLIANCE OPTIONAL ;
SQL>
SQL> SET FLAGS 'STRATEGY,INTERNAL';
SQL>
SQL> SELECT * FROM TAB1 WHERE a1 > 0;
Leaf#01 FFirst TAB1 Card=10
  BgrNdx1 NDX1 [1:0] Fan=19
0 rows selected
SQL>
SQL> SELECT * FROM TAB1 WHERE a1 > 0 OPTIMIZE USING TAB1NOTNULL;
~S: Outline "TAB1NOTNULL" used
Conjunct        Get     Retrieval sequentially of relation TAB1
0 rows selected
SQL>
SQL> INSERT INTO tab1 (a1) VALUE (11);
~Query Name: "TAB1TRIG" (action=1)
~S: Outline "TAB1TRIG" used
~S: Outline "TAB1NOTNULL" used
Conjunct        Get     Retrieval by index of relation TAB1
  Index name  NDX1 [0:0]
~S: Outline "TAB1NOTNULL" used
1 row inserted
SQL>
SQL> COMMIT;
Conjunct        Get     Retrieval sequentially of relation TAB1
```

- If the TRACE statement is activated by the RDMS$DEBUG_FLAGS "Xt" logical name or by the SET FLAGS statement, queries in the TRACE statement are merged into the query outline for the procedure. For example, the following query outline contains one query when the TRACE statement is disabled:

```
SQL> DECLARE :ln CHAR(40);
SQL>
SQL> BEGIN
cont> TRACE 'Jobs Held: ',
cont>     (SELECT COUNT(*)
cont>        FROM job_history
cont>        WHERE employee_id = '00201');
cont> SELECT last_name
cont>     INTO :ln
cont>     FROM employees
cont>     WHERE employee_id = '00201';
cont> END;
-- Oracle Rdb Generated Outline : 28-MAY-1997 16:48
create outline QO_A17FA4B41EF1A68B_00000000
id 'A17FA4B41EF1A68B966C1A0B083BFDD4'
mode 0
as (
  query (
-- Select
    subquery (
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance optional     ;
SQL>
```

If the query outline is generated with TRACE enabled, two queries appear: the first is for the subquery in the TRACE statement and the second is for the singleton SELECT statement:

```
SQL> DECLARE :ln CHAR(40);
SQL>
SQL> BEGIN
cont> TRACE 'Jobs Held: ',
cont>     (SELECT COUNT(*)
cont>        FROM job_history
cont>        WHERE employee_id = '00201');
cont> SELECT last_name
cont>     INTO :ln
cont>     FROM employees
cont>     WHERE employee_id = '00201';
cont> END;
-- Oracle  Rdb Generated Outline : 28-MAY-1997 16:48
create outline QO_A17FA4B41EF1A68B_00000000
id 'A17FA4B41EF1A68B966C1A0B083BFDD4'
mode 0
as (
  query (
-- Trace
    subquery (
      JOB_HISTORY 0   access path index       JOB_HISTORY_HASH
      )
    )
  query (
-- Select
    subquery (
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance optional     ;
~Xt: Jobs Held: 4
SQL>
```

If this second query outline is used at run time with the TRACE statement disabled, it cannot be applied to the query, as shown in the following example:

```
SQL> DECLARE :ln CHAR(40);
SQL>
SQL> BEGIN
cont> TRACE 'Jobs Held: ',
cont>     (SELECT COUNT(*)
cont>         FROM job_history
cont>         WHERE employee_id = '00201');
cont> SELECT last_name
cont>     INTO :ln
cont>     FROM employees
cont>     WHERE employee_id = '00201';
cont> END;
~S: Outline QO_A17FA4B41EF1A68B_00000000 used
~S: Outline/query mismatch; assuming JOB_HISTORY 0 renamed to EMPLOYEES 0
~S: Full compliance with the outline was not possible
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMPLOYEES_HASH [1:1]       Direct lookup
```

Because the outline was created with compliance optional, the query outline is abandoned and a new strategy is calculated. If compliance is mandatory, the query fails.

If any TRACE statement contains a subquery, Oracle Corporation recommends using two query outlines (if any are required at all) with different modes in order to run the query with and without TRACE enabled. That is, when TRACE is enabled, define RDMS$BIND_OUTLINE_MODE to match the TRACE enabled query outlines.

```
$ DEFINE RDMS$DEBUG_FLAGS "Xt"
$ DEFINE RDMS$DEBUG_FLAGS_OUTPUT TRACE.DAT
$ DEFINE RDMS$BIND_OUTLINE_MODE 10
```

Alternatively, use the SET FLAGS statement, which allows the TRACE flag to be enabled and the MODE established from within an interactive session or through dynamic SQL. This scheme allows the query to be run with TRACE enabled or disabled.

- You can use the keyword MODE to set the query outline mode from within interactive and dynamic SQL session.

```
SQL> SET FLAGS 'MODE(10),OUTLINE';
SQL> SHOW FLAGS
```

```
Alias RDB$DBHANDLE:
Flags currently set for Oracle Rdb:
   PREFIX,OUTLINE,MODE(10)
SQL> SELECT COUNT(*) FROM employees;
-- Rdb Generated Outline : 30-MAY-1997 16:35
create outline QO_B3F54F772CC05435_0000000A
id 'B3F54F772CC054350B2B454D95537995'
mode 10 as (
  query (
-- For loop
    subquery (
      subquery (
        EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
        )
      )
    )
  )
compliance optional     ;

         100
1 row selected
```

The following options are accepted:

- NOMODE - this is the same as MODE(0) and disables the display of the mode in the SHOW FLAGS statement. A mode of zero is a valid mode setting and is the default for generated query otlines.
- MODE(n) - where n can be any numeric value (positive or negative).
- MODE - the same as MODE(1)

In the previous example, the mode was set to 10 when generating the query outline. If the generated outline is added to the database, it is used only when the mode is set to 10, either by the SET FLAGS statement or by using the logical name RDMS$BIND_OUTLINE_MODE.

- Consider the following procedure, which contains a FOR loop and an UPDATE statement nested within an outer FOR loop:

```
SQL> BEGIN
cont> -- Find the employee and
cont> -- complete their current job, before being promoted
cont> FOR :cur AS EACH ROW OF CURSOR a
cont>     FOR SELECT last_name
cont>         FROM EMPLOYEES
cont>         WHERE employee_id = :emp_id
cont> DO
cont>     BEGIN
cont>     -- Display some details
cont>     TRACE 'Employee: ', :cur.last_name;
cont>
cont>     FOR :cur2 AS EACH ROW OF CURSOR b
cont>         FOR SELECT cast(job_start AS DATE ANSI) AS js,
cont>                    cast(job_end AS DATE ANSI) AS je
cont>             FROM JOB_HISTORY
cont>             WHERE employee_id = :emp_id
cont>             ORDER BY job_start
cont>     DO
cont>         TRACE ' Job Duration: ',
cont>            (COALESCE (:cur2.je, current_date) - :cur2.js) YEAR TO MONTH;
cont>     END FOR;
cont>
cont>     -- Now complete the current job
cont>     UPDATE JOB_HISTORY
cont>         SET job_end = CAST(current_date AS DATE VMS)
cont>         WHERE employee_id = :emp_id;
cont>
cont>     END;
cont> END FOR;
cont> END;
-- Oracle Rdb Generated Outline : 29-MAY-1997 22:52
create outline QO_39BBA6C4E902AB2B_00000000
id '39BBA6C4E902AB2B6A252A71A1CFFB71'
mode 0
as (
  query (
-- For loop
    subquery (
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  query (
-- For loop
    subquery (
      JOB_HISTORY 0   access path index       JOB_HISTORY_HASH
      )
    )
  query (
-- Update
    subquery (
      JOB_HISTORY 0   access path index       JOB_HISTORY_HASH
```

```
      )
    )
  )
compliance optional     ;
```

The order of the queries in the query outline represents a flattened tree structure that represents the complex execution profile of the compound statement. When extracting this tree structure, Oracle Rdb generates an order related to a bottom up representation of the optimization phase.

As a result, query outlines generated for any procedure with nested statements may appear inverted with the first table, EMPLOYEES, appearing last in the query outline.

```
-- Oracle Rdb Generated Outline : 29-MAY-1997 22:52
create outline QO_39BBA6C4E902AB2B_00000000
id '39BBA6C4E902AB2B6A252A71A1CFFB71'
mode 0
as (
  query (
-- For loop
    subquery (
      JOB_HISTORY 0   access path index       JOB_HISTORY_HASH
      )
    )
  query (
-- Update
    subquery (
      JOB_HISTORY 0   access path index       JOB_HISTORY_HASH
      )
    )
  query (
-- For loop
    subquery (
      EMPLOYEES 0     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance optional     ;
```

- When CREATE OUTLINE . . . ON TRIGGER is used then an outline for just the first compound trigger action is created. In a future release, outlines for subsequent actions will be supported.
- CREATE OUTLINE . . . ON COLUMN must reference a computed column, such as a table COMPUTED BY, AUTOMATIC or view column that contains select expressions. The CREATE will fail if no select expression is available.
- Partial outlines for view definitions may not be suitable for use in queries without providing more details in the outline.

The following example shows the outline created for the view. Note that the access path for JOB_defaults to SEQUENTIAL and therefore is not the best choice for this view. This occurs because the view normally queries with an EMPLOYEE_ID specified, which would cause the optimizer to choose index access for the JOB_HISTORY table.

```
SQL> create outline CURRENT_JOB on view CURRENT_JOB;
SQL> show outline CURRENT_JOB
     CURRENT_JOB
   Source:
   -- Rdb Generated Outline : 16-MAY-2001 15:11
   create outline CURRENT_JOB
   -- On view CURRENT_JOB
   id '9C6D98DAAF09A3E1796F7D345399028B'
   mode 0
   as (
    query (
   -- View
       subquery (
         JOB_HISTORY 0   access path sequential
           join by cross to
         EMPLOYEES 1     access path index       EMPLOYEES_HASH
         )
       )
     )
   compliance optional     ;
```

This alternate definition includes an index on JOB_HISTORY.

```
SQL> create outline CURRENT_JOB
cont>     on view CURRENT_JOB
cont> mode 0
cont> as (
cont>   query (
cont> -- View
cont>     subquery (
cont>       JOB_HISTORY 0   access path index  JH_EMPLOYEE_ID
cont>         join by cross to
cont>       EMPLOYEES 1     access path index  EMPLOYEES_HASH
cont>       )
cont>     )
cont>   )
cont> compliance optional
cont> comment is 'qo for view CURRENT_JOB';
```

The following query shows the results when applying this query outline. The table RETIRED_EMPLOYEES, as the name implies, contains all retired employees. Therefore, there should be no jobs assigned to these employees and the query should return zero rows.

```
SQL> -- should return no rows, since the employee retired and
SQL> -- there is no current job
SQL> set flags 'strategy';
SQL> select EMPLOYEE_ID
cont>  from CURRENT_JOB cj
cont>       inner join RETIRED_EMPLOYEES re
cont>       using (EMPLOYEE_ID)
cont>  where EMPLOYEE_ID = '00164';
~S: Outline "CURRENT_JOB" used
Cross block of 2 entries
Cross block entry 1
Index only retrieval of relation RETIRED_EMPLOYEES
Index name  RE_EMPLOYEE_ID [1:1]
Cross block entry 2
Cross block of 2 entries
Cross block entry 1
Conjunct
Leaf#01 FFirst JOB_HISTORY Card=274
BgrNdx1 JH_EMPLOYEE_ID [1:1] Bool Fan=17
Cross block entry 2
Conjunct        Index only retrieval of relation EMPLOYEES
Index name  EMPLOYEES_HASH [1:1]       Direct lookup
0 rows selected
SQL>
```

Note that the query outline CURRENT_JOB is reported as being used.

- 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.
- When Rdb compiles a query that references a view, it will implicitly use the view name to locate a matching query outline. This allows the database administrator to create partial query outlines that tune just that part of the query involving the view. However, if the query outline is named with the same name as a view but does not follow the structure of the view then a RDMS-F-LEVEL_MISMATCH error will be reported.

  The following example shows this problem.

```
SQL> create outline CURRENT_JOB
cont>   from (select * from CURRENT_JOB limit to 1 rows);
SQL>
SQL> show outline CURRENT_JOB;
     CURRENT_JOB
 Source:
```

```
-- Rdb Generated Outline :  2-SEP-2010 10:24
create outline CURRENT_JOB
id 'E9968EFAF723ED23DF59216A5DDE4C7D'
mode 0
as (
  query (
-- For loop
    subquery (
      subquery (
        EMPLOYEES 1     access path index       EMP_EMPLOYEE_ID
          join by match to
        JOB_HISTORY 0   access path index       JH_EMPLOYEE_ID
        )
      )
    )
  )
compliance optional     ;
SQL>
SQL> set flags 'strategy,detail(2)';
SQL>
SQL> select * from CURRENT_JOB limit to 1 rows;
~S: Outline "CURRENT_JOB" used
%RDMS-F-LEVEL_MISMATCH, the table/subquery nesting levels
in the query outline do not match the query
SQL>
```

To resolve this problem the database administrator must change the name of the outline so that it is not assumed to describe the view record selection definition.

```
SQL> create outline CURRENT_JOB_REF
cont>   from (select * from CURRENT_JOB limit to 1 rows);
SQL>
SQL> set flags 'strategy,detail(2)';
SQL>
SQL> select * from CURRENT_JOB limit to 1 rows;
~S: Outline "CURRENT_JOB_REF" used
...
 LAST_NAME        FIRST_NAME   EMPLOYEE_ID   JOB_CODE   DEPARTMENT_CODE
SUPERVISOR_ID   JOB_START
 Toliver          Alvin        00164         DMGR       MBMN
00228           21-Sep-1981
1 row selected
SQL>
SQL> select * from CURRENT_JOB where employee_id = '00164'
cont> optimize using CURRENT_JOB_REF;
~S: Outline "CURRENT_JOB_REF" used
...
 LAST_NAME        FIRST_NAME   EMPLOYEE_ID   JOB_CODE   DEPARTMENT_CODE
SUPERVISOR_ID   JOB_START
 Toliver          Alvin        00164         DMGR       MBMN
00228           21-Sep-1981
1 row selected
SQL>
```

Alternatively, create the query outline on the view itself to allow it to be used more widely.

```
SQL> create outline CURRENT_JOB
cont>   on view CURRENT_JOB;
SQL>
SQL> show outline CURRENT_JOB;
     CURRENT_JOB
 Source:
```

```
-- Rdb Generated Outline :  2-SEP-2010 10:52
create outline CURRENT_JOB
-- On view CURRENT_JOB
id '9C6D98DAAF09A3E1796F7D345399028B'
mode 0
as (
  query (
-- View
    subquery (
      EMPLOYEES 1     access path index       EMP_EMPLOYEE_ID
        join by match to
      JOB_HISTORY 0   access path index       JH_EMPLOYEE_ID
      )
    )
  )
compliance optional     ;
SQL>
SQL> set flags 'strategy,detail(2)';
SQL>
SQL> select * from CURRENT_JOB limit to 1 rows;
~S: Outline "CURRENT_JOB" used
...
SQL>
```

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     ;
```

Example 5: New Output from Query Outlines

```
SQL> BEGIN
cont> DECLARE :x INTEGER;
cont> -- Assignment
cont> SET :x = (SELECT COUNT(*) FROM TOUT_1);
cont> -- Delete statement
cont> DELETE FROM TOUT_1;
cont> -- Update statement
cont> UPDATE TOUT_1
cont>     SET a = (SELECT AVG(a) FROM TOUT_2)
cont>     WHERE a IS NULL;
cont> -- Singleton Select
cont> SELECT a INTO :x
cont>     FROM TOUT_1
cont>     WHERE a = 1;
cont> -- Trace (nothing if TRACE is disabled)
cont> TRACE 'The first value: ', (SELECT a FROM TOUT_1 LIMIT TO 1 ROW);
cont> END;
```

The query outline generated by Oracle Rdb appears with comments after the QUERY keyword in the outline.

```
-- Rdb Generated Outline : 29-MAY-1997 23:17
create outline QO_C11395E6020C6FFA_00000000
id 'C11395E6020C6FFA5A183A6CCE7C1F33'
mode 0
as (
  query (
-- Set
    subquery (
      TOUT_1 0        access path sequential
      )
    )
  query (
-- Delete
    subquery (
      TOUT_1 0        access path sequential
      )
    )
  query (
-- Update
    subquery (
      subquery (
        TOUT_2 1        access path sequential
        )
        join by cross to
      subquery (
        TOUT_1 0        access path sequential
        )
      )
    )
```

```
  query (
-- Select
    subquery (
      TOUT_1 0        access path sequential
      )
    )
  query (
-- Trace
    subquery (
      TOUT_1 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 PROFILE Statement

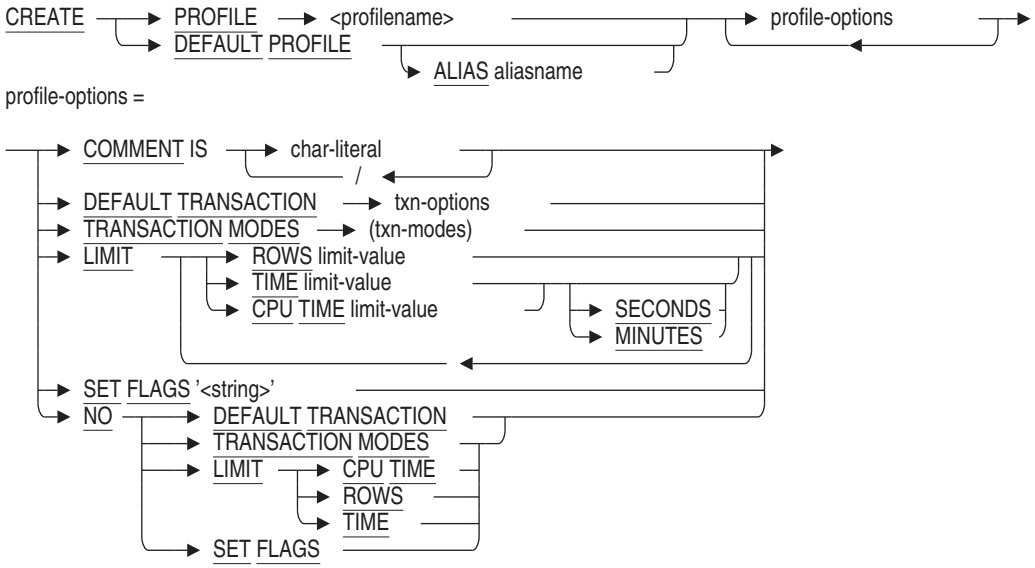
Creates a profile that extends a user definition within the database with special attributes that control transactions and resource usage. When a user attaches to the database using ATTACH, CONNECT or SET SESSION AUTHORIZATION, they will either load their assigned profile definition or inherit the default profile (if defined).

Environment

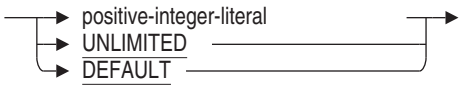
You can use the CREATE PROFILE statement:

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

Format



limit-value =



Arguments

ALIAS aliasname
When attached to multiple databases, the aliasname is required to direct the CREATE command to the appropriate database.

COMMENT IS
This optional clause can be used to add several lines of comment to the profile object. The comment is displayed by the SHOW PROFILES statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

DEFAULT PROFILE
Creates the special profile RDB$DEFAULT_PROFILE. This profile will be used by any user who is not assigned a profile using the PROFILE clause of CREATE or ALTER USER.

DEFAULT TRANSACTION
DEFAULT TRANSACTION provides a default transaction for the user. By default, Oracle Rdb starts a READ WRITE transaction if none is explicitly started. Use the DECLARE DEFAULT TRANSACTION or START DEFAULT TRANSACTION statement to make use of this definition. You can override this clause with a DECLARE, SET or START TRANSACTION statement.

Note

Oracle Rdb does not permit the RESERVING or EVALUATING clauses to appear in the default transaction.

LIMIT CPU TIME
NO LIMIT CPU TIME
LIMIT CPU TIME sets the maximum CPU time that can be used by the query compiler. The keyword DEFAULT indicates that no value is defined by this profile and is equivalent to NO LIMIT CPU TIME.

If a numeric value or the keyword UNLIMITED is specified then this value will be used even when the SET QUERY LIMIT CPU TIME statement is present in the session, or when the logical name RDMS$BIND_QG_CPU_TIMEOUT is defined.

NO LIMIT CPU TIME is the default. Units can be specified as seconds or minutes.

LIMIT ROWS
NO LIMIT ROWS
LIMIT ROWS sets the maximum number of rows that can be returned by a query started by the user. The keyword DEFAULT indicates that no value is defined by this profile and is equivalent to NO LIMIT ROWS.

If a numeric value or the keyword UNLIMITED is specified then this value will be used even when the SET QUERY LIMIT ROWS statement is present in the session, or when the logical name RDMS$BIND_QG_REC_LIMIT is defined.

NO LIMIT ROWS is the default.

LIMIT TIME
NO LIMIT TIME
LIMIT TIME sets the maximum elapsed time that can be used by the query compiler. The keyword DEFAULT indicates that no value is defined by this profile and is equivalent to NO LIMIT TIME.

If a numeric value or the keyword UNLIMITED is specified then this value will be used even when the SET QUERY LIMIT TIME statement is present in the session, or when the logical name RDMS$BIND_QG_TIMEOUT is defined.

NO LIMIT TIME is the default. Units can be specified as seconds or minutes.

SET FLAGS '<string>'
NO SET FLAGS
Specifies whether or not a database system flag is set. The value is a quoted string of keywords separated by commas. See the SET FLAGS Statement for a list of keywords that can be set.

TRANSACTION MODES
NO TRANSACTION MODES
TRANSACTION MODES provides the list of allowable transactions for this user. Please see the SET TRANSACTION MODES clause of the CREATE DATABASE and ALTER DATABASE statements for more details of txn-modes.

The transaction modes specified may include modes disabled for all database users by CREATE, IMPORT, or ALTER DATABASE statements. However, only the subset allowed by both profile and database settings will be used. For instance, if the database specifies (READ ONLY, SHARED READ, PROTECTED READ) and the profile specifies (READ ONLY, SHARED), the session will be allowed the subset (READ ONLY, SHARED READ).

Usage Notes

- It is possible to restrict the transaction modes to READ ONLY using the default profile. Use caution in this case because it is possible that no user will have READ WRITE access to undo such a definition. In this case, you can define the logical name RDMS$SET_FLAGS to the value PROFILE_OVERRIDE to allow a suitably privileged user to start a transaction without using the transaction mode restrictions in the default profile. Such a user must have database SECURITY privilege, possibly inherited from the OpenVMS SECURITY process privilege.
- The logical names RDMS$BIND_QG_REC_LIMIT, RDMS$BIND_QG_TIMEOUT, and RDMS$BIND_QG_TIMEOUT establish the process defaults for the query limit.

  The command SET QUERY LIMIT establishes the session default (unless already set by the query governor logical names).

  The profile LIMIT will either use these established defaults (LIMIT ... DEFAULT or NO LIMIT) or override them (LIMIT ... UNLIMITED or specified value).
- The string associated with the SET FLAGS clause is saved with the created profile. Any user that has this assigned profile will implicitly execute SET FLAGS during session start.

  Please notice that some SET FLAGS keywords affect actions during database attach and so have no action when defined within a profile. For example, DATABASE_PARAMETERS, generates minimal effects in such cases.

Examples

Example 1

The following example specifies the allowed transaction modes for any user assigned this profile.

```
SQL> CREATE PROFILE DECISION_SUPPORT
cont>   COMMENT IS 'limit transactions used by report writers'
cont>   TRANSACTION MODES (NO READ WRITE, READ ONLY);
```

Example 2

This example shows the use of the LIMIT clauses to set boundaries for standard database users.

```
SQL> create profile STANDARD_USER
cont>    limit rows 10000
cont>    limit time 10 minutes
cont>    limit cpu time 20 seconds;
SQL> show profile STANDARD_USER;
     STANDARD_USER
     Limit rows 10000
     Limit time 10 minutes
     Limit CPU time 20 seconds
SQL> alter profile STANDARD_USER
cont>    limit time 60 minutes;
```

Example 3

This example creates a default profile in the database. Any user who is not given an explicit profile will use this as a default.

```
SQL> create default profile
cont>      alias rdb$dbhandle
cont>      default transaction read only wait 20;
SQL> show profile
Profiles in database with filename SQL$DATABASE
     LIMIT_ROWS
     RDB$DEFAULT_PROFILE
SQL> show profile RDB$DEFAULT_PROFILE
     RDB$DEFAULT_PROFILE
     Default transaction read only wait 20
```

Example 4

The following example shows the creation of a profile with flags and the assigning of the profile to a user. When this user (SMITH) attaches to a database (ATTACH, CONNECT, DECLARE ALIAS), or uses SET SESSION AUTHORIZATION, a SET FLAGS statement will implicitly be executed using this string of keywords.

```
SQL> create profile SF_USER
cont> set flags
cont> 'old_cost_model,noindex_column_group,optimization_level(total_time)'
cont> ;
SQL>
SQL> alter user SMITH
cont> profile SF_USER;
SQL>
```

Example 5

Note that the CREATE PROFILE and ALTER PROFILE statements will validate any listed keywords in the SET FLAGs clause.

```
SQL> alter profile SF_USER
cont> set flags 'THIS_OLD_HOUSE'
cont> ;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-EXT_ERR, Oracle Rdb extension error
-RDMS-E-UNKAMBFLAG, 'THIS_OLD_HOUSE' is an unknown or ambiguous flag name
SQL>
```

Example 6

The clause NO SET FLAGS can be used to remove any flags associated with a profile. All users assigned that profile will no longer perform a SET FLAGS action during session start.

```
SQL> show profile SF_USER
     SF_USER
     Flags: "old_cost_model,noindex_column_group,optimization_level(total_time)"
SQL> alter profile SF_USER
cont> no set flags;
SQL> show  profile SF_USER
     SF_USER
SQL>
```

CREATE ROLE Statement

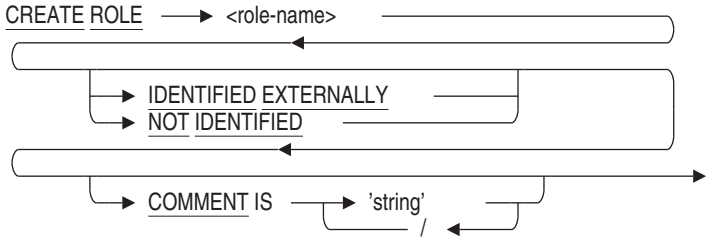
Creates a role to which privileges and other roles can be granted. A role can be granted to a user or another role. For example, you can create a role for members of a department. When a user leaves the department, the departmental role can be revoked from that user and thus exclude that user's access to the departmental data.

Environment

You can use the CREATE ROLE statement:

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

Format



Arguments

COMMENT IS 'string'
Adds a comment about the role. SQL displays the text of the comment when it executes a SHOW ROLES statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash mark (/).

IDENTIFIED EXTERNALLY
The IDENTIFIED EXERNALLY clause indicates that SQL should inherit the roles defined by the facilities of the operating system, such as rights identifiers. When a session is started, any role that is defined externally is established as part of the current user's profile.

NOT IDENTIFIED
Indicates that the role is used only with the database. The database must have SECURITY CHECKING IS INTERNAL set before using this clause.

role-name
A user-supplied name that you assign to the role. The special roles BATCH, DIALUP, INTERACTIVE, LOCAL, NETWORK, and REMOTE are reserved names that cannot be specified as a role-name.

Usage Notes

- You must have the SECURITY privilege on the database to create a role.
- The special roles BATCH, DIALUP, INTERACTIVE, LOCAL, NETWORK, and REMOTE are granted by the OpenVMS operating system when the user process is created. Therefore, these roles are reserved names and cannot be used as the role-name in the CREATE ROLE statement.
- Database roles will be created implicitly by the GRANT statement in any database set as SECURITY CHECKING IS INTERNAL if the role is not already defined and matches the name of an existing OpenVMS rights identifier.
- The role-name can be any valid SQL-name. If IDENTIFIED EXTERNALLY is used, the name must conform to OpenVMS naming conventions, that is, uppercase letters, numbers, underscore and '$' with no spaces of punctuation.
- Roles can be created for reference when SECURITY CHECKING IS EXTERNAL is set.
- If SECURITY CHECKING IS INTERNAL is set, then the GRANT statement will implicitly perform a CREATE ROLE if the role is not defined in the database and the name exists as an OpenVMS rights identifier. The following example causes both the user and role to be created.

```
SQL> grant ADMIN_USER to SMITH;
%RDB-W-META_WARN, metadata successfully updated with the reported warning
-RDMS-W-PRFCREATED, some users or roles were created
SQL> show users
Users in database with filename personnel
     SMITH
SQL> show roles
Roles in database with filename personnel
     ADMIN_USER
```

The warning message alerts the database administrator that some implicit actions were performed, but otherwise the GRANT statement was successful.

- You can display existing roles defined for a database by issuing a SHOW ROLES statement.

Examples

Example 1: Creating a Role

```
SQL> ALTER DATABASE FILENAME 'mf_personnel.rdb'
cont> SECURITY CHECKING IS INTERNAL;
SQL> ATTACH 'FILENAME mf_personnel.rdb';
SQL> CREATE ROLE WRITER;
SQL> SHOW ROLES;
Roles in database with filename mf_personnel.rdb
     WRITER
```

Example 2: Creating Roles and Granting Privileges to Those Roles

```
SQL> ALTER DATABASE FILENAME mf_personnel.rdb
cont> SECURITY CHECKING IS INTERNAL;
SQL> -- Create a role for employees in the payroll department
SQL> ATTACH 'FILENAME MF_PERSONNEL.RDB';
SQL> CREATE ROLE PAYROLL
cont> COMMENT IS 'This role allows access to various tables'
cont> /          'and procedures for use by the PAYROLL dept.';
SQL> -- Create another role for a subset of employees.
SQL> CREATE ROLE ANNUAL_LEAVE
cont> COMMENT IS 'This role is granted to PAYROLL personnel'
cont> /          'who adjust the annual leave data';
SQL> -- Grant EXECUTE privilege on module and ALL privilege on table
SQL> -- SALARY_HISTORY to all employees to whom the PAYROLL role has
SQL> -- been granted.  Grant EXECUTE privilege on module LEAVE_ADJUSTMENT
SQL> --  only to those employees who have been granted both the PAYROLL
SQL> -- and ANNUAL_LEAVE roles.
SQL> GRANT EXECUTE ON MODULE PAYROLL_UTILITIES TO PAYROLL;
SQL> GRANT ALL ON TABLE SALARY_HISTORY TO PAYROLL;
SQL> GRANT EXECUTE ON MODULE LEAVE_ADJUSTMENT
cont> to PAYROLL, ANNUAL_LEAVE;
SQL> -- User STUART joins the personnel department.  Grant him
SQL> -- the PAYROLL and ANNUAL_LEAVE roles so that he can
SQL> -- perform all functions in the payroll department.
SQL> CREATE USER STUART
cont> IDENTIFIED EXTERNALLY
SQL> GRANT PAYROLL, ANNUAL_LEAVE TO STUART;
SQL> -- User STUART is promoted to supervisor and thus
SQL> -- no longer needs access to the objects controlled by
SQL> -- the ANNUAL_LEAVE role.  Revoke that role from user
SQL> -- STUART.
SQL> REVOKE ANNUAL_LEAVE FROM STUART;
```

Example 3: Creating roles explictly using CREATE ROLE and implicitly using GRANT

This examples demonstrates creating roles that match an OpenVMS rights identifiers. The CREATE ROLE statement is used first, and then the GRANT statement. GRANT issues a warning message to alert the database administrator of the side-effect of the GRANT statement.

```
SQL> create database
cont>   filename SAMPLE
cont>   security checking is internal;
SQL> show roles;
Roles in database with filename sample
 No Roles found
SQL> create role dba_mgr identified externally;
SQL> show roles;
Roles in database with filename sample
     DBA_MGR
SQL> grant saldb_user to smith;
%RDB-W-META_WARN, metadata successfully updated with the reported warning
-RDMS-W-PRFCREATED, some users or roles were created
SQL> show roles;
Roles in database with filename sample
     DBA_MGR
     SALDB_USER
SQL>
```

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.

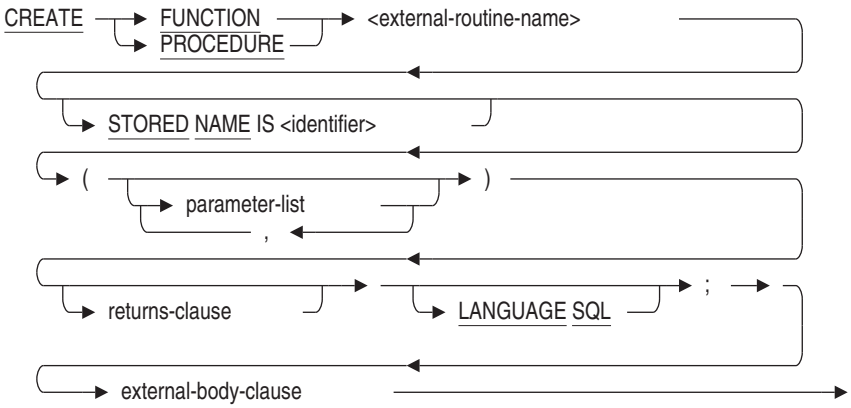
SQL can invoke an external function from anywhere you can specify a value expression. External procedures are invoked using the CALL Statement for Compound Statements.

Environment

You can use the CREATE FUNCTION and CREATE PROCEDURE statements:

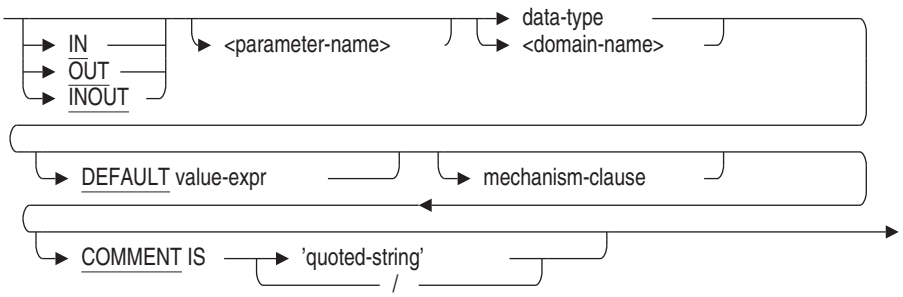
- In interactive SQL
- Embedded in host language programs to be precompiled
- As part of a procedure in an SQL module
- In dynamic SQL as a statement to be dynamically executed

Format

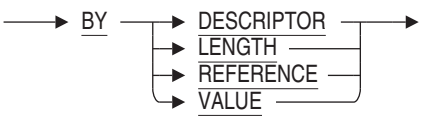


CREATE ROUTINE Statement

parameter-list =

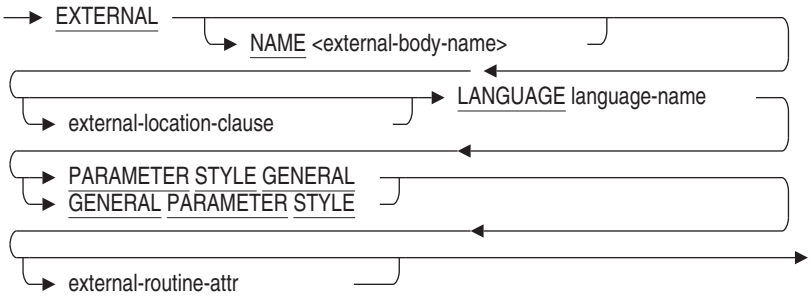


mechanism-clause =

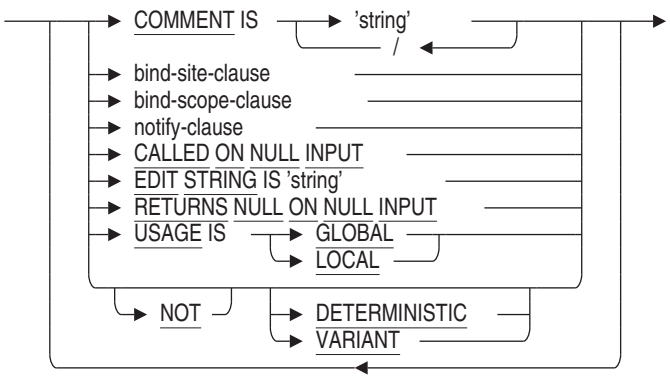


returns-clause =

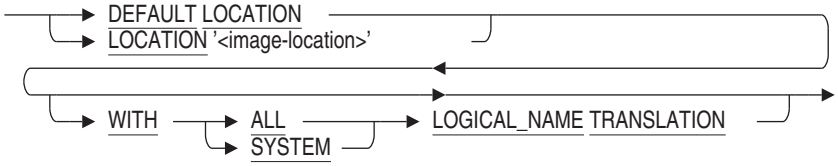
external-body-clause =



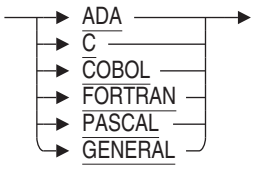
external-routine-attr =



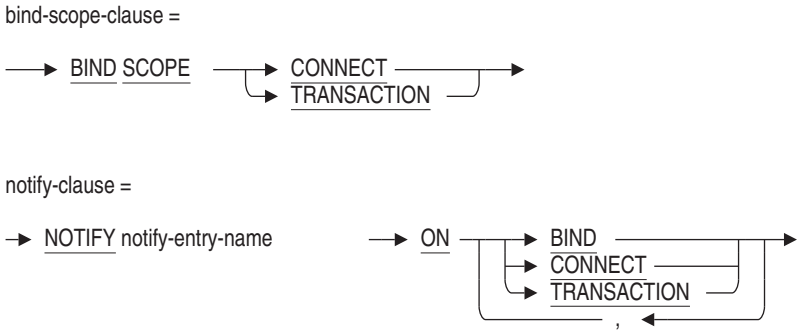
external-location-clause =



language-name =



bind-site-clause =



Arguments

BIND ON CLIENT SITE
BIND ON SERVER SITE
Selects the execution model and environment for external routine execution.

CLIENT site binding causes the external routine to be activated and executed in the OpenVMS database client (application) process. This is the default binding. This binding offers the most efficient execution characteristics, allows sharing resources such as I/O devices, and allows debugging of external routines as if they were part of the client application. However, this binding may suffer from address space limitations. Because it shares virtual memory with the database buffers, this binding is restricted to the client process system user environment, and prohibits external routine execution in cases of an application running with elevated privileges.

SERVER site binding causes the external routine to be activated in a separate process from the database client and server. The process is started on the same node at the database process. This binding offers reasonable execution characteristics, a larger address space, a true session user environment, and has no restrictions regarding client process elevated privileges. However, this binding does not permit sharing resources such as I/O devices with the client (in particular, there is no connection to the client interactive terminal), and debugging of routines is generally not possible.

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.

COMMENT IS 'string'
A description about the nature of the parameter or external routine. SQL displays the text of the comment when you execute a SHOW FUNCTION or SHOW PROCEDURE statement. Enclose the comment in single quotation marks (') and separate multiple lines in a comment with a slash (/).

DEFAULT value-expr
Specifies the default value of a parameter for a function or procedure defined with mode IN. If you omit this parameter or if the Call statement argument list or function invocation specifies the DEFAULT keyword, then the value-expr specified with this clause is used. The parameter uses NULL as the default if you do not specify a value expression explicitly.

DEFAULT LOCATION
LOCATION 'image-location'
A default or specific location for the external routine image. The resulting file specification must include the type *.exe*.

This can be an image file specification or merely a logical name.

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.

- 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.

DETERMINISTIC
NOT DETERMINISTIC

The clause controls the evaluation of an external function in the scope of a query:

- NOT DETERMINISTIC

  Specifying the NOT DETERMINISTIC 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 DETERMINISTIC clause.

- DETERMINISTIC

  Specifying the DETERMINISTIC 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 DETERMINISTIC 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 DETERMINISTIC, the function F1() may be evaluated just once depending on the optimizer. If you define the F1 function as NOT DETERMINISTIC, the function F1() is evaluated twice.

  DETERMINISTIC is the default.

The DETERMINISTIC or NOT DETERMINISTIC clause is not allowed on procedure definitions.

EDIT STRING is 'string'

Allows the value returned from the function invocation to be implicitly formatted using the EDIT STRING associated with the function. This is similar to defining an EDIT STRING on a column or domain. The EDIT STRING clause is only used by Interactive SQL.

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.

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.

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.

GENERAL PARAMETER STYLE
This is synonymous with PARAMETER STYLE GENERAL and is deprecated.

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.

LANGUAGE SQL
Names the language that calls the routine.

mechanism-clause
Defines the passing mechanism. The following list describes the passing mechanisms.

- BY DESCRIPTOR

  Allows passing character data with any parameter access mode to routines compiled by language compilers that implement the OpenVMS calling standard.

- BY LENGTH

  The LENGTH passing mechanism is the same as the DESCRIPTOR passing mechanism.

- 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.

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.

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

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 the LIST OF BYTE VARYING data type.

PARAMETER STYLE GENERAL
Passes arguments and returns values in a manner similar to the OpenVMS convention for passing arguments and returning function values.

PROCEDURE
Creates an external procedure definition.

A procedure optionally accepts a list of IN, OUT, or INOUT parameters, and is referenced by name in a CALL statement.

RETURNS result-data-type
RETURNS domain-name
Describes a 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 the LIST OF BYTE VARYING data type.

RETURNS NULL ON NULL INPUT
CALLED ON NULL INPUT
These clauses control how an external function is invoked when one or more of the function arguments is NULL. The CALLED ON NULL INPUT clause specifies that the function should be executed normally. A normal execution when the PARAMETER STYLE GENERAL clause is specified means that SQL should return a run-time error when the NULL value is detected.

The RETURNS NULL ON NULL INPUT clause instructs Oracle Rdb to avoid the function call and just return a NULL result. This option is valuable for library functions such as SIN, COS, CHECKSUM, SOUNDEX, and so on, that usually return an UNKNOWN result if an argument is NULL.

The CALLED ON NULL INPUT clause is the default.

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 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 2.2.18.

USAGE IS
Specifies how the function or procedure can be called:

- USAGE IS GLOBAL indicates that the function or procedure can be called outside the current module. This is the default.
- USAGE IS LOCAL specifies that the routine is restricted to references within the module. This clause is provided for compatibility with CREATE MODULE but is not allowed for CREATE FUNCTION or CREATE PROCEDURE.

VARIANT
NOT VARIANT
These clauses are synonyms for the DETERMINISTIC and NOT DETERMINISTIC clauses. The DETERMINISTIC clause indicates that the same inputs to the function will generate the same output. It is the same as the NOT VARIANT clause. The NOT DETERMINISTIC clause indicates that the output of the function does not depend on the inputs. It is the same as the VARIANT clause. This clause is deprecated. Use DETERMINISTIC instead.

Usage Notes

- You must have the CREATE database privilege on the database to create an external routine.
- 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 Rdb Guide to Database Design and Definition*.
- You can invoke an external procedure using the CALL statement within a compound statement.
- No more than 255 arguments may be passed to an external routine.
- Certain languages do not accept parameters passed by VALUE.
- Certain languages, such as FORTRAN, are constrained 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. 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.
- 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.
- The PARAMETER STYLE GENERAL 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 Rdb 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.
- Only one USAGE IS clause is permitted per routine definition. For external routines the clause is permitted in either the routine header, or in the external routine definition.
- If more than one routine wishes to share the same database context then they must be created together in a CREATE MODULE statement. See CREATE MODULE Statement for further details.
- If all functions and procedures are declared as USAGE IS LOCAL, then it will not be possible to execute any routines in that module. The CREATE MODULE statement will fail as shown in the following example.

```
SQL> create module M
cont>     procedure p0 (in :a integer)
cont>       usage is local;
cont>         begin
cont>         trace 'p0: ', :a;
cont>         end;
cont>         end;
cont> end module;
%SQL-W-LOCALNEVER, Local routine "P0" is never called
%SQL-F-MODALLLOCAL, Module "M" only contains local routines - invalid
module
```

Examples

Example 1: System provided integer absolute value routine

```
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>   PARAMETER STYLE GENERAL
cont>   NOT DETERMINISTIC;
SQL> --
SQL> SELECT IABS(-33) FROM JOBS LIMIT TO 1 ROW;

          33
1 row selected
```

Example 2: Using the NOT DETERMINISTIC clause, instead of the DETERMINISTIC clause

The first CREATE FUNCTION statement in the following example creates a function with the DETERMINISTIC clause. The DETERMINISTIC 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 DETERMINISTIC 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>       PARAMETER STYLE GENERAL
cont>       DETERMINISTIC;
SQL> --
SQL> -- Use a SELECT statement to pass a string literal to the function.
SQL> --
SQL> -- Because Oracle Rdb optimizes functions with the DETERMINISTIC
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$OUTPUT "HELLO"'), employee_id FROM employees
cont> LIMIT TO 5 ROWS;
HELLO
      DO_COM   EMPLOYEE_ID
           1   00164
           1   00165
           1   00166
           1   00167
           1   00168
5 rows selected
SQL> --
SQL> -- Use the NOT DETERMINISTIC clause to create the function:
SQL> --
SQL> CREATE function DO_COM (IN VARCHAR(255) BY DESCRIPTOR)
cont>       RETURNS INTEGER;
cont>       EXTERNAL NAME lib$SPAWN
cont>                LOCATION 'SYS$SHARE:LIBRTL.EXE'
cont>       LANGUAGE GENERAL
cont>       PARAMETER STYLE GENERAL
cont>       NOT DETERMINISTIC;
SQL> SELECT DO_COM('WRITE SYS$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: External function and external procedure definition

The following example demonstrates:

- 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 PARAMETER STYLE GENERAL NOT DETERMINISTIC
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 PARAMETER STYLE GENERAL
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 4: 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 5: 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
    SYMBOL_VECTOR = (CLEAR_SOUNDEX=PROCEDURE)
    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$USER:/LIBRARY, -
  CLEAR_SOUNDEX.OPT/OPT
    SYMBOL_VECTOR = (CLEAR_SOUNDEX=PROCEDURE)
    PSECT_ATTR=RDB$MESSAGE_VECTOR,NOSHR
    PSECT_ATTR=RDB$DBHANDLE,NOSHR
    PSECT_ATTR=RDB$TRANSACTION_HANDLE,NOSHR
```

Example 6: 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 7: 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 8: 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
    SYMBOL_VECTOR = (ADD_SOUNDEX_NAME=PROCEDURE)
    SYMBOL_VECTOR = (ADD_SOUNDEX_NOTIFY=PROCEDURE)
    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$USER:/LIBRARY, -
  ADD_SOUNDEX.OPT/OPT
    SYMBOL_VECTOR  = ADD_SOUNDEX_NAME
    SYMBOL_VECTOR  = ADD_SOUNDEX_NOTIFY
    PSECT_ATTR=RDB$MESSAGE_VECTOR,NOSHR
    PSECT_ATTR=RDB$DBHANDLE,NOSHR
    PSECT_ATTR=RDB$TRANSACTION_HANDLE,NOSHR
```

Example 9: 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;
```

Example 10: Using the EDIT STRING clause to add formatting to what the SQL function returns

```
SQL> create module SAMPLE
cont>     function SHOW_EMP_ID
cont>         (in :last_name varchar(30)
cont>         ,in : birthday date)
cont>     returns integer
cont>     edit string '9999"-"9999"-"99?"VOID"'
cont>     ;
cont>     return
cont>         (select cast(employee_id as integer) * 100
cont>          from EMPLOYEES
cont>          where birthday = :birthday
cont>            and last_name = :last_name);
cont> end module;
SQL>
SQL> select SHOW_EMP_ID (last_name, birthday), last_name, first_name
cont>  from EMPLOYEES
cont>  where employee_id < '00170';
               LAST_NAME        FIRST_NAME
 0000-0164-00   Toliver          Alvin
 0000-0165-00   Smith            Terry
 0000-0166-00   Dietrich         Rick
 0000-0167-00   Kilpatrick       Janet
 0000-0168-00   Nash             Norman
 0000-0169-00   Gray             Susan
6 rows selected
SQL>
SQL> -- demonstrate the output when a NULL result is returned
SQL> select SHOW_EMP_ID ('Unknown', current_date)
cont>  from EMPLOYEES
cont>  fetch first row only;

 VOID
1 row selected
SQL>
```

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 informational 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 may 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.

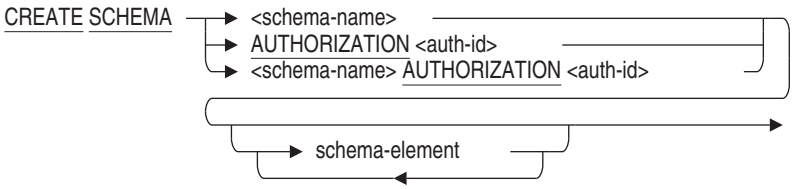
These statements require statement terminators, except when they are part of a CREATE SCHEMA or CREATE DATABASE statement. When you use these statements within a CREATE SCHEMA statement, use a statement terminator on the last schema element only. The first statement terminator that SQL encounters ends the CREATE SCHEMA statement. Later CREATE or GRANT statements are not within the scope of the CREATE SCHEMA statement.

Environment

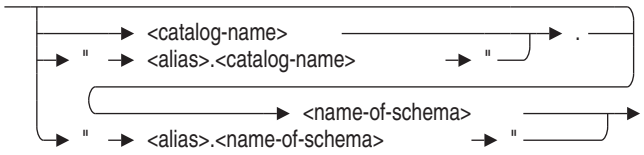
You can use the CREATE SCHEMA statement:

- 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

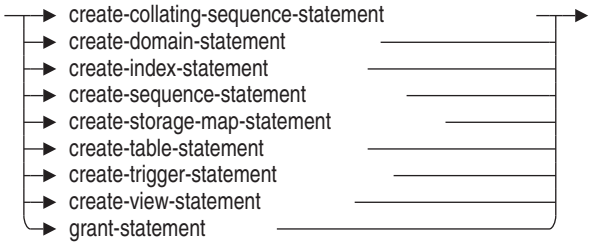
Format



schema-name =



schema-element =



Arguments

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 current ANSI/ISO SQL standard.

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-sequence-statement
See the CREATE SEQUENCE 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.

schema-element
Some CREATE statements or a GRANT statement. See the syntax diagram in this section for the complete list of allowable CREATE statements.

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 2.2.15.

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 'SQL99';
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
```

Example 2: Using a single statement to create a catalog, schema, domain, sequence and table for an existing multischema database.

```
SQL> create catalog NEW_CATALOG
cont>  create schema NEW_SCHEMA
cont>  create sequence MYSEQ
cont>  create table MYTABLE (a int identity)
cont>  create domain MONEY as integer(2)
cont> ;
SQL> show tables
User tables in database with filename abc
     NEW_CATALOG.NEW_SCHEMA.MYTABLE
SQL> show sequences
Sequences in database with filename abc
     NEW_CATALOG.NEW_SCHEMA.MYSEQ
```

```
Sequence Id: 2
Initial Value: 1
Minimum Value: 1
Maximum Value: 9223372036854775806
Next Sequence Value: 1
Increment by: 1
Cache Size: 20
No Order
No Cycle
No Randomize
Wait
    NEW_CATALOG.NEW_SCHEMA.MYTABLE

Sequence Id: 1
An identity column sequence.
Initial Value: 1
Minimum Value: 1
Maximum Value: (none)
Next Sequence Value: 1
Increment by: 1
Cache Size: 20
No Order
No Cycle
No Randomize
Wait
Comment: column IDENTITY sequence
```

Index

A

ABS
  See After-image journal (.aij) file
Account
  locking, 6–249
  unlocking, 6–249
ACL-style protection clause
  differences from ANSI/ISO-style, 6–390
ADD CACHE clause, 6–332
  of ALTER DATABASE statement, 6–26
Adding
  columns to tables, 6–199
ADD JOURNAL clause
  of ALTER DATABASE statement, 6–26
ADD STORAGE AREA clause
  See also CREATE STORAGE AREA clause
  of ALTER DATABASE statement, 6–26
ADJUSTABLE LOCK GRANULARITY clause
  of ALTER DATABASE statement, 6–27
  of CREATE DATABASE statement, 6–362
After-image journal (.aij) file, 6–43, 6–45
  allocating blocks, 6–28
  backup file, 6–32, 6–59
  backup server (ABS), 6–33
  EXTENT clause, 6–41
  fast commit processing, 6–41
  log server, 6–47
  unsuppressing the journal, 6–45
After-image journal (.aij) files
  disabling logging, 6–181
    for ALTER STORAGE MAP statement, 6–186
AIJ log server
  See ALS
ALERT OPERATOR clause
  of NOTIFY clause
    of ALTER DATABASE statement, 6–27
  options, 6–27
Alias
  for default database, 6–261
  in ATTACH statement, 6–260
  in CONNECT statement, 6–323
  in CREATE DATABASE statement, 6–362
  in CREATE DOMAIN statement, 6–425
  RDB$DBHANDLE, 6–260, 6–261, 6–323, 6–363
ALIAS clause
  of CREATE DATABASE statement, 6–362
ALL AREAS clause
  of MULTITHREAD AREA ADDITIONS clause
    of CREATE DATABASE statement, 6–363
ALLOCATION clause
  alter storage area parameter
    of ALTER DATABASE statement, 6–29
  of ALTER STORAGE AREA clause, 6–29
  of CREATE CACHE clause, 6–333
  storage area parameter
    of CREATE DATABASE statement, 6–364
ALLOCATION IS clause
  of JOURNAL clause
    of ALTER DATABASE statement, 6–28
ALS
  specifying with ALTER DATABASE statement, 6–47

ALTER CACHE clause
of ALTER DATABASE statement, 6–29
ALTER clause
of ALTER TABLE statement, 6–208
ALTER CONSTRAINT statement, 6–8
COMMENT IS clause, 6–8
RENAME TO clause, 6–9
ALTER COUNT RESULT clause
of ALTER DATABASE statement, 6–29
of CREATE DATABASE statement, 6–364
ALTER DATABASE statement, 6–14
ADD CACHE clause, 6–26
ADD JOURNAL clause, 6–26
ADD STORAGE AREA clause, 6–26
ADJUSTABLE LOCK GRANULARITY clause, 6–27
ALERT OPERATOR clause
of NOTIFY clause, 6–27
ALLOCATION clause
alter storage area parameter, 6–29
ALLOCATION IS clause
of JOURNAL clause, 6–28
ALTER CACHE clause, 6–29
ALTER COUNT RESULT clause, 6–29
altering storage area parameters, 6–30
ALTER JOURNAL clause, 6–29
ALTER STORAGE AREA clause, 6–30
ALTER TRANSACTION MODES clause, 6–31
ASYNC BATCH WRITES clause, 6–31
ASYNC PREFETCH clause, 6–32
BACKUP FILENAME clause
of JOURNAL clause, 6–32
BACKUP SERVER clause
of JOURNAL clause, 6–33
BUFFER SIZE clause, 6–33
CACHE USING clause
alter storage area parameter, 6–34
CARDINALITY COLLECTION clause, 6–34
CARRY OVER LOCKS clause, 6–34
CHECKPOINT ALL ROWS TO BACKING FILE clause, 6–36
CHECKPOINT INTERVAL clause
of FAST COMMIT clause, 6–35
CHECKPOINT TIMED clause
ALTER DATABASE statement
CHECKPOINT TIMED clause (cont'd)
of FAST COMMIT clause, 6–36
CHECKPOINT UPDATED ROWS TO BACKING FILE clause, 6–36
CHECKPOINT UPDATED ROWS TO DATABASE clause, 6–36
CHECKSUM CALCULATION clause
alter storage area parameter, 6–37
CLEAN BUFFER COUNT clause
of ASYNC BATCH WRITES clause, 6–38
COMMIT TO JOURNAL OPTIMIZATION clause
of FAST COMMIT clause, 6–38
COUNT IS clause
of ADJUSTABLE LOCK GRANULARITY clause, 6–39
DEPTH clause
of ASYNC PREFETCH clause, 6–39
of DETECTED ASYNC PREFETCH clause, 6–39
DETECTED ASYNC PREFETCH clause, 6–39
DICTIONARY clause, 6–39
DROP CACHE clause, 6–40
DROP JOURNAL clause, 6–40
DROP STORAGE AREA clause, 6–40
EXTENT clause
alter storage area parameter, 6–41
of JOURNAL clause, 6–41
FAST COMMIT clause
of JOURNAL clause, 6–41
FILENAME clause, 6–42
of ADD JOURNAL clause, 6–43
GALAXY SUPPORT clause, 6–43
GLOBAL BUFFERS clause, 6–44
INCREMENTAL BACKUP SCAN OPTIMIZATION clause, 6–44
JOURNAL clause, 6–45
JOURNAL IS UNSUPPRESSED clause
of ALTER JOURNAL clause, 6–45
LOCATION clause
of ADD CACHE clause, 6–45
of ALTER CACHE clause, 6–45
LOCKING clause

ALTER DATABASE statement
LOCKING clause (cont'd)
alter storage area parameter, 6–46
LOCK PARTITIONING clause, 6–46
LOCK TIMEOUT INTERVAL clause, 6–46
LOGMINER SUPPORT clause, 6–47
LOG SERVER clause
of JOURNAL clause, 6–47
MAXIMUM BUFFER COUNT clause
of ASYNC BATCH WRITES clause, 6–48
MEMORY ALLOCATION IS clause
of GLOBAL BUFFERS clause, 6–48
METADATA CHANGES clause, 6–48
MULTISCHEMA clause, 6–49
NO BACKUP FILENAME clause
of JOURNAL clause, 6–49
NO COMMIT TO JOURNAL OPTIMIZATION clause
of FAST COMMIT clause, 6–38
NO LOCATION clause
of ROW CACHE clause, 6–49
NO ROW CACHE clause
alter storage area parameter, 6–50
NO SWEEP INTERVAL clause
of ROW CACHE clause, 6–50
NOTIFY clause
of JOURNAL clause, 6–50
NUMBER IS clause
of GLOBAL BUFFERS clause, 6–50
NUMBER OF BUFFERS clause, 6–51
NUMBER OF CLUSTER NODES clause, 6–51
NUMBER OF RECOVERY BUFFERS clause, 6–51
NUMBER OF USERS clause, 6–52
OPEN clause, 6–52
OVERWRITE clause
of JOURNAL clause, 6–53
PAGE TRANSFER clause
of GLOBAL BUFFERS clause, 6–53
PATHNAME clause, 6–42
PRESTARTED TRANSACTIONS ARE DISABLED clause, 6–54
PRESTARTED TRANSACTIONS clause, 6–54
READ ONLY clause

ALTER DATABASE statement
READ ONLY clause (cont'd)
alter storage area parameter, 6–54
READ WRITE clause, 6–30
alter storage area parameter, 6–54
RECOVERY JOURNAL (BUFFER MEMORY IS GLOBAL) clause, 6–56
RECOVERY JOURNAL (BUFFER MEMORY IS LOCAL) clause, 6–56
RECOVERY JOURNAL (LOCATION IS) clause, 6–57
RECOVERY JOURNAL (NO LOCATION) clause, 6–57
RECOVERY JOURNAL clause, 6–57
RESERVE CACHE SLOTS clause, 6–57
RESERVE JOURNALS clause, 6–57
RESERVE SEQUENCES clause, 6–57
RESERVE STORAGE AREAS clause, 6–58
reserving slots for sequences, 6–57
restriction, 6–76, 6–77
ROW CACHE clause, 6–58
SAME BACKUP FILENAME AS JOURNAL clause
of ADD JOURNAL clause, 6–59
SECURITY CHECKING clause, 6–59
SET COUNT RESULT clause, 6–29
SET TRANSACTION MODES clause, 6–61
SHARED MEMORY clause, 6–61
SHARED MEMORY IS PROCESS RESIDENT clause, 6–397
SHUTDOWN TIME clause
of JOURNAL clause, 6–62
SNAPSHOT ALLOCATION clause
alter storage area parameter, 6–62
SNAPSHOT DISABLED clause, 6–63
SNAPSHOT ENABLED clause, 6–63
SNAPSHOT EXTENT clause
alter storage area parameter, 6–62
specifying file for checkpointed rows, 6–36
STATISTICS COLLECTION clause, 6–63
SWEEP INTERVAL clause
of ROW CACHE clause, 6–64, 6–400
SYNONYMS ARE ENABLED clause, 6–64
THRESHOLD IS clause

ALTER DATABASE statement
THRESHOLD IS clause (cont'd)
of DETECTED ASYNC PREFETCH clause, 6–64
TRANSACTION INTERVAL clause
of FAST COMMIT clause, 6–64
USER clause, 6–66
USER LIMIT clause
of GLOBAL BUFFERS clause, 6–65
USING clause
of USER clause, 6–66
WAIT clause
of OPEN clause, 6–66
WORKLOAD COLLECTION clause, 6–66
ALTER DOMAIN statement, 6–92
See also CREATE DOMAIN statement
See also DROP DOMAIN statement in Volume 3
COLLATING SEQUENCE clause, 6–96, 6–106
COMMENT IS clause, 6–97
conversion error, 6–101
domain constraint clause, 6–97
formatting clauses, 6–99
NO COLLATING SEQUENCE clause, 6–99
RENAME TO clause, 6–99
restriction, 6–101
sql-and-dtr-clause, 6–99
ALTER FUNCTION statement, 6–110
COMMENT IS clause, 6–112
COMPILE clause, 6–112
DETERMINISTIC clause, 6–113
RENAME TO clause, 6–114
RETURNS NULL clause, 6–114
VARIANT clause, 6–115
ALTER INDEX statement, 6–118
ADD PARTITION clause, 6–121
BUILD PARTITION clause, 6–134
COMMENT IS clause, 6–122
DROP PARTITION clause, 6–122
DUPLICATES ARE ALLOWED clause, 6–122
IN area-name clause, 6–122
LOGGING clause, 6–122
MAINTENANCE IS DISABLED clause, 6–123
ALTER INDEX statement (cont'd)
MAINTENANCE IS ENABLED clause, 6–123
MAINTENANCE IS ENABLED DEFERRED clause, 6–123
MAINTENANCE IS ENABLED IMMEDIATE clause, 6–123
MOVE PARTITION clause, 6–123
NODE SIZE clause, 6–124
NOLOGGING clause, 6–122
PARTITION clause, 6–124
PERCENT FILL clause, 6–124
PREFIX CARDINALITY COLLECTION IS DISABLED clause, 6–124
PREFIX CARDINALITY COLLECTION IS ENABLED clause, 6–124
PREFIX CARDINALITY COLLECTION IS ENABLED FULL clause, 6–124
REBUILD ALL PARTITIONS clause, 6–125
REBUILD PARTITION clause, 6–134
RENAME PARTITION clause, 6–125
STORE clause, 6–122
threshold clause, 6–125
TRUNCATE PARTITION clause, 6–134
USAGE UPDATE clause, 6–126
USING clause, 6–126
WITH LIMIT OF clause, 6–126
Altering
See Modifying
views, 6–252
Altering storage area parameters
of ALTER DATABASE statement, 6–30
ALTER JOURNAL clause
of ALTER DATABASE statement, 6–29
ALTER MODULE statement, 6–142
ADD clause, 6–144
COMMENT IS clause, 6–144
COMPILE clause, 6–144
DROP clause, 6–144
END MODULE clause, 6–144
RENAME TO clause, 6–144
ALTER OUTLINE statement, 6–152
COMMENT IS clause, 6–152
RENAME TO clause, 6–153

ALTER PROCEDURE statement, 6–158
COMMENT IS clause, 6–160
COMPILE clause, 6–160
RENAME TO clause, 6–162
ALTER PROFILE statement
COMMENT IS clause, 6–165
ALTER ROLE statement, 6–169
COMMENT IS clause, 6–169
IDENTIFIED EXTERNALLY clause, 6–169
NOT IDENTIFIED clause, 6–169
RENAME TO clause, 6–170
ALTER SEQUENCE statement, 6–172
CACHE clause, 6–173
COMMENT IS clause, 6–174
CYCLE clause, 6–174
DEFAULT WAIT clause, 6–178
INCREMENT BY clause, 6–174
MAXVALUE clause, 6–175
MINVALUE clause, 6–175
NOCACHE clause, 6–173
NOCYCLE clause, 6–174
NOMAXVALUE clause, 6–175
NOMINVALUE clause, 6–175
NOORDER clause, 6–176
NORANDOMIZE clause, 6–176
NOWAIT clause, 6–178
ORDER clause, 6–176
RANDOMIZE clause, 6–176
RENAME AS clause, 6–177
RESTART clause, 6–177
RESTART WITH clause, 6–177
WAIT clause, 6–178
ALTER statement
general usage notes, 6–7
ALTER STORAGE AREA clause
ALLOCATION clause, 6–29
CACHE USING clause, 6–34
CHECKSUM CALCULATION clause, 6–37
EXTENT clause, 6–41
LOCKING clause, 6–46
NO ROW CACHE clause, 6–50
of ALTER DATABASE statement, 6–30
READ ONLY clause, 6–54
READ WRITE clause, 6–54
SNAPSHOT ALLOCATION clause, 6–62
ALTER STORAGE AREA clause (cont'd)
SNAPSHOT EXTENT clause, 6–62
ALTER STORAGE MAP statement, 6–181
COMMENT IS clause, 6–184
COMPILE clause, 6–184
COMPRESSION clause, 6–185
disabling logging to .aij file, 6–181
LOGGING clause, 6–186
NOLOGGING clause, 6–186
NO PLACEMENT VIA INDEX clause, 6–186
PARTITIONING IS NOT UPDATABLE clause, 6–187
PARTITIONING IS UPDATABLE clause, 6–186
PARTITION name clause, 6–186
PLACEMENT VIA INDEX clause, 6–187
RENAME PARTITION clause, 6–187
REORGANIZE clause, 6–187
STORAGE MAP clause, 6–187
STORE LISTS clause, 6–188
THRESHOLDS clause, 6–188
ALTER SYNONYM statement, 6–197
ALTER TABLE statement, 6–199
AFTER COLUMN clause, 6–207
ALTER clause, 6–208
AUTOMATIC clause, 6–208
AUTOMATIC INSERT clause, 6–208
AUTOMATIC UPDATE clause, 6–208
BEFORE COLUMN clause, 6–207
constraint-attributes clause, 6–210
DEFERRABLE clause, 6–210
disable clause, 6–215
DROP CONSTRAINT clause, 6–214
enable clause, 6–215
GENERATED ALWAYS AS (value-expr) clause, 6–217
GENERATED ALWAYS AS IDENTITY clause, 6–216
GENERATED BY DEFAULT AS (value-expr) clause, 6–218
GENERATED BY DEFAULT AS IDENTITY clause, 6–217
IDENTITY clause, 6–218
INITIALLY DEFERRED clause, 6–210
large memory is disabled, 6–219

ALTER TABLE statement (cont'd)
large memory is enabled, 6–219
modifying tables, 6–199
NOT DEFERRABLE clause, 6–210
NULL column constraint, 6–219
REFERENCES clause, 6–220
RENAME TO clause, 6–220
specifying domains for data types, 6–419
ALTER TRANSACTION MODES clause
CREATE DATABASE statement, 6–364
of ALTER DATABASE statement, 6–31
transaction modes, 6–65, 6–402
ALTER TRIGGER statement, 6–245
COMMENT IS clause, 6–245, 6–246
COMPILE clause, 6–245
DISABLE clause, 6–245
ENABLE clause, 6–246
RENAME TO clause, 6–246
ALTER USER statement, 6–249
ACCOUNT LOCK clause, 6–249
ACCOUNT UNLOCK clause, 6–249
COMMENT IS clause, 6–250
IDENTIFIED EXTERNALLY clause, 6–250
PROFILE clause, 6–250
PUBLIC clause, 6–250
RENAME TO clause, 6–250
ALTER VIEW statement, 6–252
rules for modifying views, 6–253
ANSI/ISO-style protection clause
differences from ACL-style, 6–390
AREAS keyword
of REORGANIZE clause, 6–184
AS clause
of CONNECT statement, 6–323
Assigning row caches, 6–365
ASYNC BATCH WRITES clause
of ALTER DATABASE statement, 6–31
of CREATE DATABASE statement, 6–364
Asynchronous batch-write, 6–31, 6–364
Asynchronous prefetch, 6–32, 6–365
ASYNC PREFETCH clause
of ALTER DATABASE statement, 6–32
of CREATE DATABASE statement, 6–365
ATOMIC keyword
in compound statement, 6–305
Attaching to a database
multiple attachments to same database, 6–266
with ATTACH statement, 6–258 to 6–268
Attach specifications
in ATTACH statement, 6–263
in CONNECT statement, 6–324
ATTACH statement
attach specifications, 6–263
database option, 6–262
DBKEY SCOPE clause, 6–262
default alias, 6–260
FILENAME clause, 6–263
MULTISCHEMA IS ON clause, 6–264
PATHNAME clause, 6–264
PRESTARTED TRANSACTIONS clause, 6–264
repository path names, 6–264
RESTRICTED ACCESS clause, 6–265
ROWID SCOPE clause, 6–265
Authentication
external, 6–250
user, 6–45, 6–263, 6–324, 6–326, 6–379
Auth-id
See Authorization identifier
AUTHORIZATION clause
of CREATE SCHEMA statement, 6–552
Authorization identifier
in CREATE SCHEMA statement, 6–552
AUTOMATIC clause, 6–208

B

Backup
incremental, 6–44, 6–377
BACKUP FILENAME clause
of JOURNAL clause
of ALTER DATABASE statement, 6–32
options, 6–32
BACKUP SERVER clause
of JOURNAL clause
of ALTER DATABASE statement, 6–33

BEGIN DECLARE statement, 6–269, 6–270
  required terminators, 6–269
BEGIN keyword
  in compound statement, 6–306
Beginning label
  in compound statement, 6–306
Block
  compound statements, 6–306
B-tree index
  See Sorted index
Buffer
  for asynchronous batch writes, 6–38, 6–48, 6–368, 6–381
  for asynchronous prefetch, 6–39, 6–373
  for detected asynchronous prefetch, 6–39, 6–373
BUFFER SIZE clause
  of ALTER DATABASE statement, 6–33
  of CREATE DATABASE statement, 6–365

C

Cached rows
  specifying a checkpoint interval, 6–367
CACHE SIZE clause
  of CREATE CACHE clause, 6–334
CACHE USING clause
  alter storage area parameter
    of ALTER DATABASE statement, 6–34
  of ALTER STORAGE AREA clause, 6–34
  storage area parameter
    of CREATE DATABASE statement, 6–365
CALL statement
  compound, 6–275
  simple, 6–272
CARDINALITY COLLECTION clause
  of ALTER DATABASE statement, 6–34
  of CREATE DATABASE statement, 6–366
Carry-over lock optimization, 6–34, 6–366
CARRY OVER LOCKS clause
  of ALTER DATABASE statement, 6–34
  of CREATE DATABASE statement, 6–366
CASE (searched) control statement, 6–279
  ELSE compound-use-statement, 6–279
  THEN compound-use-statement, 6–279
  WHEN predicate, 6–280
CASE (simple) control statement
  ELSE clause, 6–282
  of compound statement, 6–281
  THEN clause, 6–282
  WHEN clause, 6–282
Catalog
  adding comments on, 6–286
  CREATE CATALOG statement, 6–344
  creating, 6–344
CATALOG clause
  of CONNECT statement, 6–323
Character data type
  in CREATE DOMAIN statement, 6–424
Character set
  creating databases with, 6–353
  database
    default, 6–354, 6–372
    identifier, 6–354
    national, 6–354
  for database, 6–377, 6–383
  using
    CREATE DOMAIN statement with, 6–419
    multiple with CREATE DATABASE statement, 6–353
CHECKPOINT
  of CREATE CACHE clause, 6–334
Checkpoint interval, 6–41
  specifying for cached rows, 6–367
CHECKPOINT INTERVAL clause
  of FAST COMMIT clause
    of ALTER DATABASE statement, 6–35
Checkpoint record, 6–35
CHECKPOINT TIMED clause
  of FAST COMMIT clause
    of ALTER DATABASE statement, 6–36
  of ROW CACHE clause
    of CREATE DATABASE statement, 6–367

CHECKSUM CALCULATION clause
alter storage area parameter
of ALTER DATABASE statement, 6–37
of ALTER STORAGE AREA clause, 6–37
storage area parameter
of CREATE DATABASE statement, 6–368
CLEAN BUFFER COUNT clause
of ASYNC BATCH WRITES clause
of ALTER DATABASE statement, 6–38
of CREATE DATABASE statement, 6–368
CLOSE statement, 6–284
Closing a cursor, 6–284
Collating sequence
See also COLLATING SEQUENCE clause, CREATE COLLATING SEQUENCE statement, DROP COLLATING SEQUENCE statement, NO COLLATING SEQUENCE clause
ALTER DOMAIN statement, 6–106
altering, 6–92
CREATE COLLATING SEQUENCE statement
in CREATE DATABASE statement, 6–370
restriction, 6–350, 6–405
creating, 6–348
in CREATE SCHEMA statement, 6–552
on index fields, 6–452
restriction, 6–351
COLLATING SEQUENCE clause
of ALTER DOMAIN statement, 6–96
of CREATE DATABASE statement, 6–369
of CREATE DOMAIN statement, 6–424
Column
adding comments on, 6–286
adding to tables, 6–199
automatic, 6–208
default value, 6–214
defining, 6–199
deleting from tables, 6–199
display order, 6–207
modifying in tables, 6–199
Column constraints
CHECK, 6–209
NOT NULL, 6–219
NULL, 6–219
PRIMARY KEY, 6–219
UNIQUE, 6–221
Column default value, 6–212, 6–224, 6–233
COMMENT clause
of COLLATING SEQUENCE clause
of CREATE DATABASE statement, 6–369
COMMENT ON statement, 6–286
in CREATE DATABASE statement
restriction, 6–407
Comments
adding to catalogs, 6–286
adding to columns, 6–286
adding to constraints, 6–286
adding to databases, 6–286
adding to domains, 6–286
adding to indexes, 6–286
adding to index storage map definition, 6–122
adding to outlines, 6–286, 6–493
adding to profiles, 6–286
adding to roles, 6–524
adding to schemas, 6–286
adding to synonyms, 6–286
adding to tables, 6–286
multiple, 6–289
specifying for a user definition, 6–250
COMMIT statement
writing changes to a database, 6–294 to 6–300
COMMIT TO JOURNAL OPTIMIZATION clause
of FAST COMMIT clause
of ALTER DATABASE statement, 6–38
Compound statements, 6–301, 6–314
assigning a name to, 6–308
beginning label in, 6–306
block, 6–306
CASE (searched) control statement, 6–279
CASE (simple) control statement, 6–281
for-counted-loop clause, 6–307
lock-table-statement clause, 6–308
naming query outline for, 6–309

Compound statements (cont'd)
OPTIMIZE AS clause, 6–308
OPTIMIZE OUTLINE clause, 6–308
OPTIMIZE USING clause, 6–309
OPTIMIZE WITH clause, 6–309
PRAGMA clause, 6–309
restriction, 6–306, 6–312
searched-case-statement clause, 6–306
simple-case-statement clause, 6–310
while-statement clause, 6–311
WITH HOLD clause, 6–311
Compound-use statement
in compound statement, 6–301
Compressing
integer column values for indexes, 6–442
key suffixes for indexes, 6–435, 6–445
COMPRESSION clause
of ALTER STORAGE MAP statement, 6–185
Configuration file
authentication information, 6–263
Configuration parameter
SQL_PASSWORD, 6–263
SQL_USERNAME, 6–263
Connecting to a database
with ATTACH statement, 6–258 to 6–268
Connection, 6–318
Connection name, 6–318
CONNECT statement, 6–318
AS clause, 6–323
attach specifications, 6–324
CATALOG clause, 6–323
default alias, 6–323
FILENAME clause, 6–324
NAMES clause, 6–324
PATHNAME clause, 6–324
repository path names, 6–324
SCHEMA clause, 6–325
TO clause, 6–325
Constraint
adding to tables, 6–199
DEFERRABLE clause, 6–210
deleting from tables, 6–199
domain
adding, 6–97
altering, 6–97
Constraint
domain (cont'd)
creating, 6–425
naming in
ALTER VIEW statement, 6–253
NOT DEFERRABLE clause, 6–210
Constraint attributes
DEFERRABLE, 6–210
DEFERRABLE INITIALLY DEFERRED, 6–210
DEFERRABLE INITIALLY IMMEDIATE, 6–210
INITIALLY DEFERRED, 6–210
INITIALLY IMMEDIATE, 6–210
INITIALLY IMMEDIATE DEFERRABLE, 6–210
NOT DEFERRABLE, 6–210
NOT DEFERRABLE INITIALLY IMMEDIATE, 6–210
Constraints
disabling, 6–215
disabling validation of, 6–216
specifying evaluation time, 6–210
Control statements
CASE (searched), 6–279
CASE (simple), 6–281
in compound statement, 6–301
Conversion
errors converting
domains, 6–101
COUNT IS clause
of ADJUSTABLE LOCK GRANULARITY clause
of ALTER DATABASE statement, 6–39
of CREATE DATABASE statement, 6–369
CREATE CACHE clause, 6–332
ALLOCATION clause, 6–333
CACHE SIZE clause, 6–334
CHECKPOINT clause, 6–334
EXTENT clause, 6–334
LARGE MEMORY clause, 6–335
LOCATION clause, 6–335
NO LOCATION clause, 6–335
NUMBER OF RESERVED ROWS clause, 6–335

CREATE CACHE clause (cont'd)
NUMBER OF SWEEP ROWS clause, 6–336
of CREATE DATABASE statement, 6–332, 6–369
ROW LENGTH clause, 6–336
ROW REPLACEMENT clause, 6–336
ROW SNAPSHOT clause, 6–337
SHARED MEMORY clause, 6–337
CREATE CACHE statement
SHARED MEMORY IS PROCESS RESIDENT clause, 6–337
CREATE CATALOG statement, 6–344, 6–345
of CREATE DATABASE statement, 6–369
CREATE COLLATING SEQUENCE statement, 6–348, 6–351
in CREATE DATABASE statement restriction, 6–350
in CREATE SCHEMA statement, 6–552
of CREATE DATABASE statement, 6–370
STORED NAME IS clause, 6–349
CREATE DATABASE clause
SNAPSHOT ALLOCATION clause, 6–397
CREATE DATABASE statement, 6–353
ADJUSTABLE LOCK GRANULARITY clause, 6–362
ALIAS clause, 6–362
ALL AREAS clause
of MULTITHREAD AREA ADDITIONS clause, 6–363
ALLOCATION clause
storage area parameter, 6–364
ALTER COUNT RESULT clause, 6–364
ALTER TRANSACTION MODES clause, 6–364
ASYNC BATCH WRITES clause, 6–364
ASYNC PREFETCH clause, 6–365
BUFFER SIZE clause, 6–365
CACHE USING clause
storage area parameter, 6–365
CARDINALITY COLLECTION clause, 6–366
CARRY OVER LOCKS clause, 6–366
CHECKPOINT ALL ROWS TO BACKING FILE clause, 6–367
CHECKPOINT TIMED clause
of ROW CACHE clause, 6–367
CREATE DATABASE statement (cont'd)
CHECKPOINT TIMED EVERY n SECONDS clause, 6–367
CHECKPOINT UPDATED ROWS TO BACKING FILE clause, 6–367
CHECKPOINT UPDATED ROWS TO DATABASE clause, 6–367
CHECKSUM CALCULATION clause
storage area parameter, 6–368
CLEAN BUFFER COUNT clause
of ASYNC BATCH WRITES clause, 6–368
COLLATING SEQUENCE clause, 6–369
collating sequence restriction, 6–405
COMMENT clause
of COLLATING SEQUENCE clause, 6–369
COUNT IS clause
of ADJUSTABLE LOCK GRANULARITY clause, 6–369
CREATE CACHE clause, 6–332, 6–369
CREATE CATALOG statement, 6–369
CREATE COLLATING SEQUENCE statement, 6–370
CREATE DOMAIN statement, 6–370
CREATE FUNCTION statement, 6–370
CREATE INDEX statement, 6–370
CREATE MODULE statement, 6–370
CREATE PROCEDRUE statement, 6–370
CREATE SCHEMA statement, 6–371
CREATE SEQUENCE statement, 6–371
CREATE STORAGE AREA clause, 6–371
CREATE STORAGE MAP statement, 6–371
CREATE TABLE statement, 6–371
CREATE TRIGGER statement, 6–371
CREATE VIEW statement, 6–371
DBKEY SCOPE clause, 6–371
default alias, 6–363
default character set, 6–372
DEFAULT STORAGE AREA clause, 6–372
DEPTH clause
of ASYNC PREFETCH clause, 6–373
of DETECTED ASYNC PREFETCH clause, 6–373
DETECTED ASYNC PREFETCH clause, 6–374

CREATE DATABASE statement (cont'd)
DICTIONARY clause, 6–374
environment, 6–354
EXTENT clause
storage area parameter, 6–374
FILENAME clause, 6–375
GALAXY SUPPORT clause, 6–376
GLOBAL BUFFERS clause, 6–376
GRANT statement, 6–377
identifier character set, 6–377
including COMMENT ON statement
restriction, 6–407
INCREMENTAL BACKUP SCAN
OPTIMIZATION clause, 6–377
in dynamic SQL, 6–354
in embedded SQL, 6–354
in interactive SQL, 6–354
INTERVAL clause
storage area parameter, 6–377
LIMIT TO AREAS clause
of MULTITHREAD AREA ADDITIONS
clause, 6–378
LIST STORAGE AREA clause, 6–378
LOCATION IS clause
of ROW CACHE clause, 6–379
LOCKING clause
storage area parameter, 6–380
LOCK PARTITIONING clause, 6–379
LOCK TIMEOUT INTERVAL clause, 6–380
LOGMINER SUPPORT clause, 6–381
MAXIMUM BUFFER COUNT clause
of ASYNC BATCH WRITES clause,
6–381
MEMORY ALLOCATION IS clause
of GLOBAL BUFFERS clause, 6–382
METADATA CHANGES clause, 6–382
MULTISCHEMA clause, 6–382
MULTITHREAD AREA ADDITIONS clause,
6–383
national character set, 6–383
NCS options
of COLLATING SEQUENCE clause,
6–383
NO LOCATION clause
of ROW CACHE clause, 6–384
NO ROW CACHE clause

CREATE DATABASE statement
NO ROW CACHE clause (cont'd)
storage area parameter, 6–384
NOTIFY clause
of JOURNAL clause, 6–384
NUMBER IS clause
of GLOBAL BUFFERS clause, 6–385,
6–388, 6–401
NUMBER OF BUFFERS clause, 6–385
NUMBER OF CLUSTER NODES clause,
6–386
NUMBER OF RECOVERY BUFFERS clause,
6–386
NUMBER OF USERS clause, 6–387
OPEN clause, 6–387
PAGE FORMAT clause
storage area parameter, 6–387
PAGE SIZE clause
storage area parameter, 6–388
PATHNAME clause, 6–389
PRESTARTED TRANSACTIONS clause,
6–390
PROTECTION clause, 6–390
RECOVERY JOURNAL (BUFFER MEMORY
IS GLOBAL) clause, 6–391
RECOVERY JOURNAL (BUFFER MEMORY
IS LOCAL) clause, 6–391
RECOVERY JOURNAL clause, 6–392
repository path names, 6–389
RESERVE CACHE SLOTS clause, 6–392
RESERVE JOURNALS clause, 6–392
RESERVE n SEQUENCES clause, 6–393
RESERVE STORAGE AREAS clause, 6–393
reserving slots for sequences, 6–393
RESTRICTED ACCESS clause, 6–394
restriction, 6–405
root file parameters, 6–394
ROW CACHE clause, 6–394
ROWID SCOPE clause, 6–395
schemas, 6–353
SECURITY CHECKING clause, 6–395
SEGMENTED STRING clause, 6–396
SET COUNT RESULT clause, 6–364
SET TRANSACTION MODES clause, 6–396
SHARED MEMORY clause, 6–397
SNAPSHOT ALLOCATION clause

CREATE DATABASE statement
SNAPSHOT ALLOCATION clause (cont'd)
alter storage area parameter, 6–397
SNAPSHOT CHECKSUM ALLOCATION clause, 6–397
SNAPSHOT DISABLED clause, 6–398
SNAPSHOT ENABLED clause, 6–397, 6–399
SNAPSHOT EXTENT clause
storage area parameter, 6–398
SNAPSHOT FILENAME clause
storage area parameter, 6–398
specifying a checkpoint interval, 6–367
specifying file for checkpointed rows, 6–367
STATISTICS COLLECTION clause, 6–399
storage area parameters, 6–399
SYSTEM INDEX parameter, 6–400
SYSTEM INDEX PREFIX CARDINALITY COLLECTION parameter, 6–400
SYSTEM INDEX PREFIX TYPE IS parameter, 6–400
THRESHOLD IS clause
of DETECTED ASYNC PREFETCH clause, 6–401
THRESHOLDS clause
storage area parameter, 6–401
used in program
restriction, 6–407
USER clause, 6–401
USING clause
of USER clause, 6–402
using multiple character sets, 6–353
WAIT clause
of OPEN clause, 6–403
WORKLOAD COLLECTION clause, 6–403
CREATE DOMAIN statement, 6–419
COLLATING SEQUENCE clause, 6–424, 6–431
COMMENT IS clause, 6–425
domain constraint clause, 6–425
FROM path-name clause, 6–426
in CREATE SCHEMA statement, 6–552
NO COLLATING SEQUENCE clause, 6–427
of CREATE DATABASE statement, 6–370
STORED NAME IS clause, 6–428
use of National Character Set (NCS) utility, 6–424
CREATE FUNCTION statement, 6–434, 6–529
in CREATE SCHEMA statement, 6–552
of CREATE DATABASE statement, 6–370
RETURNS NULL clause, 6–537
CREATE INDEX statement, 6–435
DISABLE COMPRESSION clause, 6–440
ENABLE COMPRESSION clause, 6–440
in CREATE SCHEMA statement, 6–552
MAINTENANCE clause, 6–442
of CREATE DATABASE statement, 6–370
PREFIX CARDINALITY COLLECTION IS DISABLED clause, 6–444
PREFIX CARDINALITY COLLECTION IS ENABLED clause, 6–444
PREFIX CARDINALITY COLLECTION IS ENABLED FULL clause, 6–444
restriction, 6–454
STORED NAME IS clause, 6–446
TYPE IS HASHED ORDERED clause, 6–447
TYPE IS HASHED SCATTERED clause, 6–447
TYPE IS SORTED clause, 6–449
TYPE IS SORTED RANKED clause, 6–450
CREATE MODULE statement, 6–464
in CREATE SCHEMA statement, 6–552
of CREATE DATABASE statement, 6–370
CREATE OUTLINE statement, 6–490
CREATE PROCEDURE statement, 6–517, 6–529
in CREATE SCHEMA statement, 6–552
of CREATE DATABASE statement, 6–370
Create Routine statement
See CREATE FUNCTION statement
See CREATE PROCEDURE statement
CREATE ROUTINE statement
in CREATE SCHEMA statement, 6–552
of CREATE DATABASE statement, 6–370
CREATE SCHEMA statement, 6–550
See also CREATE DATABASE statement
AUTHORIZATION clause, 6–552
authorization identifier, 6–552
CREATE COLLATING SEQUENCE statement, 6–552
CREATE DOMAIN statement, 6–552
CREATE FUNCTION statement, 6–552

CREATE SCHEMA statement (cont'd)
CREATE INDEX statement, 6–552
CREATE MODULE statement, 6–552
CREATE PROCEDURE statement, 6–552
CREATE SEQUENCE statement, 6–552
CREATE STORAGE MAP statement, 6–553
CREATE TABLE statement, 6–553
CREATE TRIGGER statement, 6–553
CREATE VIEW statement, 6–553
environment, 6–551
GRANT statement, 6–553
in dynamic SQL, 6–551
in embedded SQL, 6–551
in interactive SQL, 6–551
of CREATE DATABASE statement, 6–371
schema name clause, 6–553
CREATE SEQUENCE statement
in CREATE SCHEMA statement, 6–552
CREATE statement
general usage notes, 6–331
CREATE STORAGE AREA clause
of CREATE DATABASE statement, 6–371
CREATE STORAGE MAP statement
in CREATE SCHEMA statement, 6–553
of CREATE DATABASE statement, 6–371
CREATE TABLE statement
in CREATE SCHEMA statement, 6–553
of CREATE DATABASE statement, 6–371
specifying domains for data types, 6–419
CREATE TRIGGER statement
in CREATE SCHEMA statement, 6–553
of CREATE DATABASE statement, 6–371
CREATE VIEW statement
in CREATE SCHEMA statement, 6–553
of CREATE DATABASE statement, 6–371
Creating
catalogs, 6–344
collating sequence, 6–348
restriction, 6–350, 6–405
comments, 6–286
constraints
in ALTER TABLE statement, 6–199
databases, 6–353
collating sequence restriction, 6–350, 6–405
Creating
databases (cont'd)
using multiple character sets, 6–353
domains, 6–419
indexes, 6–435
query outline, 6–490
row cache, 6–332
schemas, 6–550
storage areas, 6–26
Cursor
closing, 6–284

D

Data
inserting into rows automatically, 6–208
Database
access, 6–52, 6–66
adding comments on, 6–286
allocating buffers, 6–51, 6–385
allocating pages, 6–29, 6–364
allocating snapshot pages, 6–62, 6–397
ALTER DATABASE statement, 6–14
assigning row caches, 6–365
attaching to
with ATTACH statement, 6–258 to 6–268
automatic recovery process, 6–51, 6–386
buffer size, 6–33, 6–365
creating, 6–353
default, 6–363
default character set, 6–354, 6–372
disabling snapshot file, 6–63, 6–398
enabling global buffers, 6–44, 6–376
enabling multischema for, 6–264
enabling snapshot file, 6–63, 6–397, 6–399
identifier character set, 6–354, 6–377
invoking, 6–258 to 6–268
limiting users, 6–52, 6–387
maximum number of cluster nodes, 6–51, 6–386
multifile, 6–353
multiple attachments to same database, 6–266
national character set, 6–354, 6–383
page size, 6–388
restricted access to, 6–265, 6–394

Database (cont'd)
single-file, 6–353
specifying extent pages, 6–62, 6–398
specifying maximum number of global buffers, 6–65, 6–401
specifying memory allocation, 6–48, 6–382
specifying multischema attribute for, 6–49, 6–382
specifying number of global buffers, 6–50, 6–385
Database access
restricted, 6–265, 6–394
Database default storage area, 6–372
Database environment
attaching database to, 6–258
implicit, 6–318
Database key
in hashed index, 6–448
Database option
for OpenVMS, 6–262
of ATTACH statement, 6–262
Databases
disabling user access to, 6–249
Database statistics
collection of, 6–63, 6–399
Data definition
disabling changes, 6–48, 6–382
enabling changes, 6–48, 6–382
Date-time data types
in CREATE DOMAIN statement, 6–97, 6–425
DBKEY SCOPE clause
of ATTACH statement, 6–262
of CREATE DATABASE statement, 6–371
Deadlock
on multiple attachments to same database, 6–266
Deassigning row cache, 6–384
DECdtm services, 6–327
Declaring variable
in compound statement, 6–314
Default character set
of CREATE DATABASE statement, 6–372
of database, 6–354, 6–372
Default database
with ATTACH statement, 6–260
with CREATE DATABASE statement, 6–363
DEFAULT STORAGE AREA clause
of CREATE DATABASE statement, 6–372
Default value
dropping
in ALTER DOMAIN statement, 6–98
in ALTER TABLE statement, 6–214
removing, 6–101, 6–224
specifying
in ALTER DOMAIN statement, 6–101, 6–105
in ALTER TABLE statement, 6–212, 6–220, 6–224, 6–233
in CREATE DOMAIN statement, 6–97, 6–425, 6–428, 6–431
DEFERRABLE clause
of ALTER TABLE statement, 6–210
DEFERRABLE constraint attribute, 6–210
DEFERRABLE INITIALLY DEFERRED constraint attribute, 6–210
DEFERRABLE INITIALLY IMMEDIATE constraint attribute, 6–210
Deleting
columns in tables, 6–199
constraints in tables, 6–199
DEPTH clause
of ASYNC PREFETCH clause
of ALTER DATABASE statement, 6–39
of CREATE DATABASE statement, 6–373
of DETECTED ASYNC PREFETCH clause
of ALTER DATABASE statement, 6–39
of CREATE DATABASE statement, 6–373
Detected asynchronous prefetch, 6–39, 6–374
DETECTED ASYNC PREFETCH clause
of ALTER DATABASE statement, 6–39
of CREATE DATABASE statement, 6–374
DICTIONARY clause
of ALTER DATABASE statement, 6–39
of CREATE DATABASE statement, 6–374

DISABLE COMPRESSION clause
of CREATE INDEX statement, 6–440
Disabling data definition changes, 6–48, 6–382
Disabling index, 6–123
Disabling index compression, 6–440
DISK keyword
of PAGE TRANSFER clause, 6–388
Distributed transaction manager, 6–327
Domain
See also ALTER DOMAIN statement; CREATE DOMAIN statement; DROP DOMAIN statement in Volume 3
adding comments on, 6–286
creating
using character set, 6–419
default value, 6–97, 6–98, 6–101, 6–105, 6–425, 6–428, 6–431
specifying
in ALTER TABLE statements, 6–419
in CREATE TABLE statements, 6–419
in SQL module parameter declarations, 6–419
instead of data type, 6–419
Domain constraint
adding, 6–97
altering, 6–97
creating, 6–425
Domain constraint clause
of ALTER DOMAIN statement, 6–97
of CREATE DOMAIN statement, 6–425
Domains
adding comments to, 6–97, 6–425
DROP CACHE clause
of ALTER DATABASE statement, 6–40
DROP CONSTRAINT clause
of ALTER TABLE statement, 6–214
DROP JOURNAL clause
of ALTER DATABASE statement, 6–40
Dropping
after-image journal, 6–40
row cache, 6–40
storage area, 6–40
DROP STORAGE AREA clause
of ALTER DATABASE statement, 6–40

E

ENABLE COMPRESSION clause
of CREATE INDEX statement, 6–440
Enabling data definition changes, 6–48, 6–382
Enabling index compression, 6–440
END DECLARE statement
required terminators, 6–269
Ending
transactions
COMMIT statement, 6–294 to 6–300
Ending label
in compound statement, 6–307
END keyword
in compound statement, 6–307
Entry
modifying for a user class, 6–249
modifying for a user name, 6–249
Environments
See Database environment
Evaluation time
specifying for constraints, 6–210
EXTENT clause
alter storage area parameter
of ALTER DATABASE statement, 6–41
of ALTER STORAGE AREA clause, 6–41
of CREATE CACHE clause, 6–334
of JOURNAL clause
ALTER DATABASE statement, 6–41
storage area parameter
of CREATE DATABASE statement, 6–374
Extent page, specifying, 6–398
External functions
creating, 6–434, 6–529
EXTERNAL NAME IS clause
See STORED NAME IS clause
External procedure
calling, 6–275
creating, 6–517, 6–529
External routine
creating, 6–529

F

FAST COMMIT clause
  of JOURNAL clause
    of ALTER DATABASE statement, 6–41
Fast commit processing, 6–41
FILENAME clause
  ATTACH statement, 6–263
  CONNECT statement, 6–324
  of ADD JOURNAL clause
    of ALTER DATABASE statement, 6–43
  of ALTER DATABASE statement, 6–42
  of CREATE DATABASE statement, 6–375
File specification
  in ALTER DATABASE statement, 6–42
  in CREATE DATABASE statement, 6–375
Filling storage areas, 6–185
Formatting clauses
  using in ALTER DOMAIN statement, 6–99
  using in CREATE DOMAIN statement, 6–427
FROM path-name clause
  in CREATE DOMAIN statement, 6–426

G

GALAXY SUPPORT clause
  of ALTER DATABASE statement, 6–43
  of CREATE DATABASE statement, 6–376
Global buffers
  enabling, 6–44, 6–376
  specifying default number of buffers, 6–50, 6–385
  specifying size of global buffer pool, 6–48, 6–382
GLOBAL BUFFERS clause
  of ALTER DATABASE statement, 6–44
  of CREATE DATABASE statement, 6–376
Global field
  See Domain
GRANT statement
  in CREATE SCHEMA statement, 6–553
  of CREATE DATABASE statement, 6–377

H

Hash bucket, 6–448
Hashed index, 6–448
  adding partition to, 6–121
  compressing, 6–440

I

Identifier character set
  of CREATE DATABASE statement, 6–377
  of database, 6–354, 6–377
Incremental backup, 6–44, 6–377
INCREMENTAL BACKUP SCAN OPTIMIZATION clause
  of ALTER DATABASE statement, 6–44
  of CREATE DATABASE statement, 6–377
Index
  adding comments on, 6–286
  ALTER INDEX statement, 6–118
  associating directly with a storage area, 6–122
  compressed, 6–442
  CREATE INDEX statement, 6–435
  defining, 6–435
  disabling, 6–123
  disabling compression, 6–440
  enabling, 6–123
  enabling compression, 6–440
  hashed, 6–448
  limiting key characters, 6–445
  naming a partition of, 6–124
  partitioned, 6–441
  restrictions on field attribute and data type, 6–452
  sorted nonranked, 6–449
  sorted ranked, 6–450
  thresholds for logical areas, 6–446
Index compression
  compression octets, 6–439
  disabled, 6–440
  enabled, 6–440
  hashed, 6–440
  minimum length, 6–443
  restriction, 6–128, 6–440, 6–441

Index compression (cont'd)
sorted, 6–440
Indexes
altering, 6–122
converting to non-unique, 6–122
disabling logging to .aij file, 6–122
Index key
specifying limits for, 6–126
Index storage map definition
adding a comment to, 6–122
INITIALLY DEFERRED clause
of ALTER TABLE statement, 6–210
INITIALLY DEFERRED constraint attribute, 6–210
INITIALLY DEFERRED DEFERRABLE constraint attribute, 6–210
INITIALLY IMMEDIATE constraint attribute, 6–210
INITIALLY IMMEDIATE DEFERRABLE constraint attribute, 6–210
INITIALLY IMMEDIATE NOT DEFERRABLE constraint attribute, 6–210
INITIALLY IMMEDIATE NOT DEFERRED constraint attribute, 6–210
INOUT parameter
for stored function, 6–469
IN parameter
for stored function, 6–469
Internationalization features
See also COLLATING SEQUENCE clause
See also CREATE COLLATING SEQUENCE statement
See also NO COLLATING SEQUENCE clause
CREATE DATABASE statement, COLLATING SEQUENCE clause, 6–369
CREATE DATABASE statement, CREATE COLLATING SEQUENCE clause, 6–370
CREATE SCHEMA statement, CREATE COLLATING SEQUENCE statement, 6–552
INTERVAL clause
storage area parameter
of CREATE DATABASE statement, 6–377

J

JOURNAL clause
of ALTER DATABASE statement, 6–45
Journal fast commit, 6–41
Journaling
adding journal file, 6–78
after-image journals, 6–45
backup server, 6–33
extensible, 6–78
JOURNAL IS UNSUPPRESSED clause
of ALTER JOURNAL clause
of ALTER JOURNAL clause, 6–45
Journal reservation, 6–57, 6–392
restriction, 6–57

L

Label
beginning, 6–306
Labeled compound statement, 6–306
LARGE MEMORY clause
of CREATE CACHE clause, 6–335
Limiting number of database users, 6–52, 6–387
Limits and parameters
maximum index size, 6–443
maximum length for an index key, 6–454
maximum length of comment string, 6–291
maximum length of database object name, 6–497
maximum number of buffer blocks, 6–33
maximum number of users, 6–52, 6–387
maximum row length for row caches, 6–336
maximum snapshot extent pages, 6–48
minimum block allocation for .aij file, 6–28
minimum number of users, 6–52, 6–387
minimum snapshot extent pages, 6–49
LIMIT TO AREAS clause
of MULTITHREAD AREA ADDITIONS clause
of CREATE DATABASE statement, 6–378
List
filling storage areas
randomly, 6–185

List
filling storage areas (cont'd)
sequentially, 6–185
storing in multiple storage areas, 6–188
LIST STORAGE AREA clause
of CREATE DATABASE statement, 6–378
LOCATION clause
of ADD CACHE clause
of ALTER DATABASE statement, 6–45
of ALTER CACHE clause
of ALTER DATABASE statement, 6–45
of CREATE CACHE clause, 6–335
LOCATION IS clause
of ROW CACHE clause
of CREATE DATABASE statement, 6–379
LOCKING clause
alter storage area parameter
ALTER DATABASE statement, 6–46
of ALTER STORAGE AREA clause, 6–46
storage area parameter
of CREATE DATABASE statement, 6–380
Lock optimization, 6–34, 6–366
LOCK PARTITIONING clause
of ALTER DATABASE statement, 6–46
of CREATE DATABASE statement, 6–379
LOCK TIMEOUT INTERVAL clause
of ALTER DATABASE statement, 6–46
of CREATE DATABASE statement, 6–380
Lock timeouts, 6–46, 6–380
Logging to after-image journal (.aij) file
disabling for ALTER STORAGE MAP statement, 6–186
Logical area threshold
for indexes, 6–446, 6–458
LOGMINER SUPPORT clause
of ALTER DATABASE statement, 6–47
of CREATE DATABASE statement, 6–381
Log server
See ALS
LOG SERVER clause
of JOURNAL clause
of ALTER DATABASE statement, 6–47

M

MAINTENANCE IS DISABLED clause
ALTER INDEX statement, 6–123
MAINTENANCE IS ENSABLED clause
ALTER INDEX statement, 6–123
MAXIMUM BUFFER COUNT clause
of ASYNC BATCH WRITES clause
of ALTER DATABASE statement, 6–48
of CREATE DATABASE statement, 6–381
MEMORY ALLOCATION IS clause
of GLOBAL BUFFERS clause
of ALTER DATABASE statement, 6–48
of CREATE DATABASE statement, 6–382
MEMORY keyword
of PAGE TRANSFER clause, 6–388
METADATA CHANGES clause
of ALTER DATABASE statement, 6–48
of CREATE DATABASE statement, 6–382
Mixed storage areas, 6–388
Modifying
column definitions, 6–199
columns in tables, 6–199
comments, 6–286
indexes, 6–118
storage maps, 6–181
table definitions, 6–199
Modules
adding comments to, 6–144
Multifile databases, 6–353
Multinational character set
See OpenVMS National Character Set (NCS) utility
Multiple attachments to same database, 6–266
Multiple character sets
using with CREATE DATABASE statement, 6–353
MULTISCHEMA clause
of ALTER DATABASE statement, 6–49
of CREATE DATABASE statement, 6–382

Multischema databases
attaching to, 6–264
creating, 6–49, 6–382
MULTISCHEMA IS ON clause
in ATTACH statement, 6–264
Multistatement procedure
compound statement and, 6–301
restriction, 6–312
Multistring comment
ALTER MODULE statement, 6–144
COMMENT ON statement, 6–289
CREATE COLLATING SEQUENCE statement, 6–349
CREATE DATABASE statement, 6–369
CREATE FUNCTION statement, 6–533
CREATE MODULE statement, 6–467
CREATE OUTLINE statement, 6–493
CREATE PROCEDURE statement, 6–533
MULTITHREAD AREA ADDITIONS clause
of CREATE DATABASE statement, 6–383

N

Named compound statement, 6–306
NAMES clause
of CONNECT statement, 6–324
National character set
of CREATE DATABASE statement, 6–383
of database, 6–354, 6–383
National Character Set (NCS) utility
See OpenVMS National Character Set (NCS) utility
NCS options
of COLLATING SEQUENCE clause
of CREATE DATABASE statement, 6–383
NO BACKUP FILENAME clause
of JOURNAL clause
of ALTER DATABASE statement, 6–49
NO COLLATING SEQUENCE clause
of ALTER DOMAIN statement, 6–99
of CREATE DOMAIN statement, 6–427
NO COMMIT TO JOURNAL OPTIMIZATION clause
of FAST COMMIT clause
of ALTER DATABASE statement, 6–38
NO LOCATION clause
of CREATE CACHE clause, 6–335
of ROW CACHE clause
of ALTER DATABASE statement, 6–49
of CREATE DATABASE statement, 6–384
Nonranked B-tree index
See Sorted index
Nonstored procedures
calling stored procedures from, 6–272, 6–275
NO PLACEMENT VIA INDEX clause
ALTER STORAGE MAP statement, 6–186
NO ROW CACHE clause
alter storage area parameter
of ALTER DATABASE statement, 6–50
of ALTER STORAGE AREA clause, 6–50
storage area parameter
of CREATE DATABASE statement, 6–384
NO SWEEP INTERVAL clause
of ROW CACHE clause
of ALTER DATABASE statement, 6–50
NOT ATOMIC keyword
in compound statement, 6–305
NOT DEFERRABLE clause
of ALTER TABLE statement, 6–210
NOT DEFERRABLE constraint attribute, 6–210
NOT DEFERRABLE INITIALLY IMMEDIATE constraint attribute, 6–210
NOTIFY clause
of JOURNAL clause
of ALTER DATABASE statement, 6–50
of CREATE DATABASE statement, 6–384
NULL column constraint, 6–219
NUMBER IS clause
of GLOBAL BUFFERS clause
of ALTER DATABASE statement, 6–50
of CREATE DATABASE statement, 6–385, 6–388, 6–401

NUMBER OF BUFFERS clause
of ALTER DATABASE statement, 6–51
of CREATE DATABASE statement, 6–385
NUMBER OF CLUSTER NODES clause
of ALTER DATABASE statement, 6–51
of CREATE DATABASE statement, 6–386
NUMBER OF RECOVERY BUFFERS clause
of ALTER DATABASE statement, 6–51
of CREATE DATABASE statement, 6–386
NUMBER OF RESERVED ROWS clause
of CREATE CACHE clause, 6–335
NUMBER OF SWEEP ROWS clause
of CREATE CACHE clause, 6–336
NUMBER OF USERS clause
of ALTER DATABASE statement, 6–52
of CREATE DATABASE statement, 6–387

O

ON ALIAS keywords
in compound statement, 6–308
Online modification
of database, 6–71
of storage areas, 6–79
OPEN clause
of ALTER DATABASE statement, 6–52
of CREATE DATABASE statement, 6–387
WAIT clause, 6–66
OpenVMS National Character Set (NCS) utility
See also NCS options
default library, 6–349, 6–383
SYS$LIBRARY:NCS$LIBRARY, 6–349, 6–383
used by ALTER DOMAIN statement, 6–96
used by CREATE COLLATING SEQUENCE statement, 6–348
used by CREATE DATABASE statement, 6–383
used by CREATE DOMAIN statement, 6–424
Operator class
notified of journaling event, 6–27
Optimizer
restrictions on index fields, 6–452
Optimizing
NOWAIT lock acquisition, 6–34, 6–366
Oracle Rdb databases
specifying in ATTACH statement, 6–262
OTHERWISE clause
in index definition, 6–453
of ALTER STORAGE MAP statement, 6–190
Outine
adding comments on, 6–286
OUT parameter
for stored function, 6–469
Overflow partition
for storage map, 6–190
in index definition, 6–453
OVERWRITE clause
of JOURNAL clause
of ALTER DATABASE statement, 6–53

P

PAGE FORMAT clause
storage area parameter
of CREATE DATABASE statement, 6–387
PAGE keyword
of REORGANIZE clause, 6–186
Page-level locking, 6–380
PAGE SIZE clause
storage area parameter
of CREATE DATABASE statement, 6–388
PAGE TRANSFER clause
DISK keyword, 6–53
MEMORY keyword, 6–53
of GLOBAL BUFFERS clause
of ALTER DATABASE statement, 6–53
Parameter
database root file, 6–353
for stored function, 6–469
storage area, 6–353, 6–399
partitioned indexes
specifying limits for, 6–126
Partitioning indexes, 6–441
PARTITIONING IS NOT UPDATABLE clause
ALTER STORAGE MAP statement, 6–187

PARTITIONING IS UPDATABLE clause
ALTER STORAGE MAP statement, 6–186
Partitions
adding to hashed index, 6–121
dropping from an index, 6–122
moving, 6–123
naming, 6–186
renaming, 6–125
PATHNAME clause
ATTACH statement, 6–264
CONNECT statement, 6–324
of ALTER DATABASE statement, 6–42
of CREATE DATABASE statement, 6–389
Performance
improving database automatic recovery process, 6–51, 6–386
PLACEMENT VIA INDEX clause
of ALTER STORAGE MAP statement, 6–187
Prefetch
asynchronous, 6–32, 6–365
detected asynchronous, 6–39, 6–374
PREFIX CARDINALITY COLLECTION, 6–124, 6–400
PREFIX CARDINALITY COLLECTION IS DISABLED clause
ALTER INDEX statement, 6–124
CREATE INDEX statement, 6–444
PREFIX CARDINALITY COLLECTION IS ENABLED clause
ALTER INDEX statement, 6–124
CREATE INDEX statement, 6–444
PREFIX CARDINALITY COLLECTION IS ENABLED FULL clause
ALTER INDEX statement, 6–124
CREATE INDEX statement, 6–444
Prestarted transaction
disabling, 6–264, 6–390
PRESTARTED TRANSACTIONS ARE DISABLED clause
of ALTER DATABASE statement, 6–54
PRESTARTED TRANSACTIONS clause
of ALTER DATABASE statement, 6–54
of ATTACH statement, 6–264
of CREATE DATABASE statement, 6–390
Primary key values
altering the sequence for generating, 6–172
Privilege
PROTECTION clause
of CREATE DATABASE statement, 6–390
Process
failure, 6–51, 6–386
Profile
adding comments on, 6–286
PROTECTION clause
of CREATE DATABASE statement, 6–390
PUBLIC user, 6–250

Q

Query outlines
creating, 6–490
naming for a compound statement, 6–309

R

Ranked B-tree index
See Sorted index
RDB$CLIENT_DEFAULTS.DAT configuration file, 6–263
RDB$DBHANDLE default alias, 6–261
in ATTACH statement, 6–260
in CONNECT statement, 6–323
in CREATE DATABASE statement, 6–363
RDB$SYSTEM storage area
changing to read/write, 6–30, 6–54
changing to read-only, 6–30, 6–54
restriction, 6–77
Rdb/VMS databases
See Oracle Rdb databases
Read/write RDB$SYSTEM storage area
changing, 6–30, 6–54
creating, 6–30, 6–54
Read/write storage area
changing, 6–54
creating, 6–54
Read/write system table
changing, 6–54
creating, 6–54

READ ONLY clause
alter storage area parameter
of ALTER DATABASE statement, 6–54
of ALTER STORAGE AREA clause, 6–54
Read-only RDB$SYSTEM storage area
changing, 6–54
creating, 6–54
Read-only storage area
changing, 6–54
creating, 6–54
Read-only system table
changing, 6–54
creating, 6–54
READ WRITE clause
alter storage area parameter
of ALTER DATABASE statement, 6–54
of ALTER STORAGE AREA clause, 6–54
READ WRITE clause of ALTER DATABASE statement, 6–30
Recovery buffers
specifying in
ALTER DATABASE statement, 6–51
CREATE DATABASE statement, 6–386
RECOVERY JOURNAL clause
of ALTER DATABASE statement, 6–57
of CREATE DATABASE statement, 6–392
Recovery-unit journal file, 6–51, 6–386
REFERENCES clause
of ALTER TABLE statement, 6–220
Referencing table, 6–216, 6–220
RENAME PARTITION clause
of ALTER STORAGE MAP statement, 6–187
REORGANIZE clause
of ALTER STORAGE MAP statement, 6–187
Repository
path names
in ATTACH statement, 6–264
in CONNECT statement, 6–324
in CREATE DATABASE statement, 6–389
in CREATE DOMAIN statement, 6–426
RESERVE CACHE SLOTS clause
of ALTER DATABASE statement, 6–57
of CREATE DATABASE statement, 6–392
RESERVE JOURNALS clause
of ALTER DATABASE statement, 6–57
of CREATE DATABASE statement, 6–392
RESERVE SEQUENCES clause
of ALTER DATABASE statement, 6–57
RESERVE STORAGE AREAS clause
of ALTER DATABASE statement, 6–58
of CREATE DATABASE statement, 6–393
Reserving
after-image journals, 6–392
after-image journal slots, 6–57
row cache slots, 6–57, 6–392
storage area slots
restriction, 6–79
storage areas slots, 6–58, 6–393
RESTRICTED ACCESS clause
of ATTACH statement, 6–265
of CREATE DATABASE statement, 6–394
Restricted access to database, 6–265, 6–394
Restriction
ALTER DATABASE statement, 6–76, 6–77, 6–79
ALTER DOMAIN statement, 6–101
compound statement, 6–312
CREATE DATABASE statement, 6–405
including COMMENT ON statement, 6–407
used in program, 6–407
CREATE DOMAIN statement
repository field structures, 6–429
for row cache, 6–79, 6–338
in compound statement, 6–306
index compression, 6–128, 6–440, 6–441
index naming, 6–454
multistatement procedures, 6–312
Norwegian collating sequence, 6–351
one alias referenced in a compound statement, 6–301, 6–312
RDB$SYSTEM storage area, 6–77
reserving journals, 6–57
reserving storage areas, 6–79
single-file database, 6–76, 6–405
snapshot file name, 6–76, 6–405

Roles
adding comments to, 6–165, 6–169, 6–245
inheriting from operating system facilities, 6–169
renaming, 6–170
Root file parameters
of CREATE DATABASE statement, 6–353, 6–394
Row cache
adding, 6–26
altering, 6–29
assigning to storage area, 6–34
assignment, 6–365
backing store directory for, 6–45, 6–49, 6–379, 6–384
creating, 6–332
deassignment, 6–384
default backing store directory for, 6–45, 6–49, 6–379, 6–384
dropping, 6–40
enabling, 6–58, 6–394
specifying a directory, 6–45
ROW CACHE clause
of ALTER DATABASE statement, 6–58
of CREATE DATABASE statement, 6–394
Row cache reservation, 6–57, 6–392
Row caches
checkpointing and, 6–36, 6–367
specifying file for checkpointed rows, 6–36, 6–367
specifying where checkpointed rows are written, 6–36, 6–367
ROWID SCOPE clause
of ATTACH statement, 6–265
of CREATE DATABASE statement, 6–395
ROW LENGTH clause
of CREATE CACHE clause, 6–336
Row-level locking, 6–380
ROW REPLACEMENT clause
of CREATE CACHE clause, 6–336
Rows
inserting data into automatically, 6–208
ROW SNAPSHOT clause
of CREATE CACHE clause, 6–337
Ruj file
disabling logging to ALTER STORAGE MAP statement, 6–181
RUJ file
specifying device and directory for, 6–57
specifying global or local memory allocation, 6–56, 6–391

S

SAME BACKUP FILENAME AS JOURNAL clause
of ADD JOURNAL clause
of ALTER DATABASE statement, 6–59
Schema
See also Database
adding comments on, 6–286
CREATE DATABASE statement, 6–353
CREATE SCHEMA statement, 6–550
defining, 6–550
naming, 6–553
SCHEMA clause
of CONNECT statement, 6–325
Security checking
external, 6–59, 6–395
internal, 6–59, 6–395
Security profile
changing for a user name, 6–250
Segmented string
whose segments vary greatly in size, 6–378
SEGMENTED STRING clause
of CREATE DATABASE statement, 6–396
SEGMENTED STRING STORAGE AREA clause
of CREATE DATABASE statement, 6–378
Sequences
adding comments to, 6–174
altering, 6–172
ascending, 6–174
cycling of, 6–174
descending, 6–174
keeping values in memory, 6–173
maximum value, 6–175
minimum value, 6–175
ordered, 6–176
preallocating values for, 6–173

Sequences (cont'd)
randomizing, 6–176
reserving slots for, 6–57, 6–393
specifying increment of, 6–174
specifying the wait state, 6–178
unordered, 6–176
SET control statement, 6–314
SET TRANSACTION MODES clause
of ALTER DATABASE statement, 6–61
of CREATE DATABASE statement, 6–396
transaction modes, 6–65, 6–402
SHARED MEMORY clause
of ALTER DATABASE statement, 6–61
of CREATE CACHE clause, 6–337
of CREATE DATABASE statement, 6–397
Shared system buffers (SSB), 6–61, 6–337, 6–397
SHUTDOWN TIME clause
of JOURNAL clause
of ALTER DATABASE statement, 6–62
Single-file database, 6–353
restriction, 6–76, 6–405
Slots
reserving for sequences, 6–57, 6–393
SNAPSHOT ALLOCATION clause
alter storage area parameter
of ALTER DATABASE statement, 6–62
of CREATE DATABASE statement, 6–397
of ALTER STORAGE AREA clause, 6–62
of CREATE DATABASE clause, 6–397
SNAPSHOT CHECKSUM ALLOCATION clause
alter storage area parameter
of CREATE DATABASE statement, 6–397
SNAPSHOT DISABLED clause
of ALTER DATABASE statement, 6–63
of CREATE DATABASE statement, 6–398
SNAPSHOT ENABLED clause
of ALTER DATABASE statement, 6–63
of CREATE DATABASE statement, 6–397, 6–399
SNAPSHOT EXTENT clause
alter storage area parameter
of ALTER DATABASE statement, 6–62
SNAPSHOT EXTENT clause (cont'd)
of ALTER STORAGE AREA clause, 6–62
storage area parameter
of CREATE DATABASE statement, 6–398
Snapshot file
moving, 6–77
Snapshot file name
restriction, 6–76, 6–405
SNAPSHOT FILENAME clause
storage area parameter
of CREATE DATABASE statement, 6–398
snapshots of database records, 6–337
Sorted index
compressing, 6–440
nonranked, 6–449
ranked, 6–450
SPAM thresholds, 6–446
SQL mapping routine
ALTER STORAGE MAP statement, 6–184
SQL module language
specifying domains for data types, 6–419
SQL statements
ALTER ROLE, 6–169
ALTER SEQUENCE, 6–172
ALTER TRIGGER, 6–245
ALTER USER, 6–249
CASE (searched) control, 6–279
SQL_PASSWORD configuration parameter, 6–263
SQL_USERNAME configuration parameter, 6–263
SSB
See Shared system buffers
Statistics
disabling for a database, 6–63, 6–399
Statistics collection
database, 6–63, 6–399
STATISTICS COLLECTION clause
of ALTER DATABASE statement, 6–63
of CREATE DATABASE statement, 6–399
Storage area
adding, 6–26, 6–58, 6–78

Storage area (cont'd)
ADD STORAGE AREA clause of ALTER DATABASE statement, 6–26
ALTER STORAGE AREA clause of ALTER DATABASE statement, 6–30
creating asynchronously, 6–383
database default, 6–372
DROP STORAGE AREA clause of ALTER DATABASE statement, 6–40
for lists, 6–188
filling randomly, 6–185
filling sequentially, 6–185
for segmented strings, 6–378
mixed, 6–388
moving, 6–77
uniform, 6–387
Storage area parameter, 6–353
allocating pages, 6–29, 6–364
allocating snapshot pages, 6–62, 6–397
assigning row cache, 6–365
cache assignment, 6–34
checksum calculation, 6–37, 6–368
data pages, 6–377
deassigning row cache, 6–384
extent options, 6–41, 6–374
locking level, 6–46, 6–380
of CREATE DATABASE statement, 6–399
page format, 6–387
page size, 6–388
read/write options, 6–54
snapshot extent, 6–398
snapshot file name, 6–398
specifying extent pages, 6–62
thresholds, 6–401
Storage area reservation, 6–58, 6–393
STORAGE MAP clause
of ALTER STORAGE MAP statement, 6–187
Storage maps
adding a comment to, 6–184
altering a comment, 6–184
ALTER STORAGE MAP statement, 6–181
modifying, 6–181
STORE clause
of ALTER STORAGE MAP statement, 6–187
Stored function
See also CREATE MODULE statement
creating, 6–464
parameters, 6–469
Stored module
creating, 6–464
STORED NAME IS clause
of CREATE COLLATING SEQUENCE statement, 6–349
of CREATE DOMAIN statement, 6–428
of CREATE INDEX statement, 6–446
Stored procedure
See also CREATE MODULE statement
calling, 6–272, 6–275
creating, 6–464
Stored routine
See Stored function
See Stored procedure
STORE LISTS clause
of ALTER STORAGE MAP statement, 6–188
SWEEP INTERVAL clause
of ROW CACHE clause
of ALTER DATABASE statement, 6–64, 6–400
Synonym
adding comments on, 6–286
SYS$LIBRARY:NCS$LIBRARY
default NCS library, 6–349, 6–383
System failure, 6–51, 6–386
SYSTEM INDEX parameter
of CREATE DATABASE statement, 6–400
SYSTEM INDEX PREFIX CARDINALITY COLLECTION parameter
of CREATE DATABASE statement, 6–400
SYSTEM INDEX PREFIX TYPE IS parameter
of CREATE DATABASE statement, 6–400
System tables
Consult online SQL Help for this information
SYSTEM_USER function
setting, 6–66, 6–265, 6–401

T

Table
  adding
    columns, 6–199
    constraints, 6–199
  adding comments on, 6–286
  ALTER TABLE statement, 6–199
  deleting
    columns, 6–199
    constraints, 6–199
  modifying columns, 6–199
  referencing, 6–216, 6–220
Table columns
  adding, 6–199
Tables
  disabling constraints for, 6–215
  disabling triggers for, 6–215
  displayed order of columns for, 6–207
Terminators
  required for BEGIN DECLARE statement, 6–269
  required for END DECLARE statement, 6–269
THRESHOLD IS clause
  of DETECTED ASYNC PREFETCH clause
    of ALTER DATABASE statement, 6–64
    of CREATE DATABASE statement, 6–401
THRESHOLDS clause
  of ALTER STORAGE MAP statement, 6–188
  of CREATE INDEX statement, 6–446
  storage area parameter
    of CREATE DATABASE statement, 6–401
Timeout intervals
  specifying, 6–46, 6–380
TO clause
  of CONNECT statement, 6–325
TRANSACTION INTERVAL clause
  of FAST COMMIT clause
    of ALTER DATABASE statement, 6–64
Transaction modes
  for ALTER TRANSACTION MODES clause, 6–65, 6–402
  for SET TRANSACTION MODES clause, 6–65, 6–402
Transactions
  COMMIT statement, 6–294 to 6–300
  CONNECT statement, 6–318
  prestarted
    disabling, 6–264, 6–390
Transferring pages
  to disk, 6–53, 6–388
  via memory, 6–53, 6–388
Triggers
  compiling, 6–245
  disabling, 6–215, 6–245
  enabling, 6–245, 6–246
TYPE IS HASHED ORDERED clause
  of CREATE INDEX statement, 6–447
TYPE IS HASHED SCATTERED clause
  of CREATE INDEX statement, 6–447
TYPE IS SORTED clause
  of CREATE INDEX statement, 6–449
TYPE IS SORTED RANKED clause
  of CREATE INDEX statement, 6–450

U

Undo/redo recovery processing
  checkpointing, 6–41
Uniform storage areas, 6–387
  threshold values, 6–446
Unique indexes
  converting to non-unique, 6–122
User authentication
  ALTER DATABASE statement, 6–45
  ATTACH statement, 6–263
  CONNECT statement, 6–324, 6–326
  CREATE DATABASE statement, 6–379
User class
  modifying entry for, 6–249
USER clause
  ATTACH statement, 6–265, 6–268
  CONNECT statement, 6–325
  of ALTER DATABASE statement, 6–66

USER clause (cont'd)
of CREATE DATABASE statement, 6–401
USER LIMIT clause
of GLOBAL BUFFERS clause
of ALTER DATABASE statement, 6–65
User name
changing, 6–250
changing security profile for, 6–250
modifying entry for, 6–249
Users
authentication of, 6–250
defining a comment for, 6–250
disabling access to database, 6–249
User session
See Connection
USING clause
ATTACH statement, 6–266
CONNECT statement, 6–325
of USER clause
of ALTER DATABASE statement, 6–66
of CREATE DATABASE statement, 6–402

V

Value expression
DEFAULT value, 6–97, 6–99, 6–212, 6–468, 6–469, 6–533
VARCHAR data type
index field, 6–452
Very large memory (VLM), 6–335
View
altering, 6–252

W

WAIT clause
of OPEN clause
of ALTER DATABASE statement, 6–66
of CREATE DATABASE statement, 6–403
Wait state
specifying for sequences, 6–178
WORKLOAD COLLECTION clause
of ALTER DATABASE statement, 6–66
of CREATE DATABASE statement, 6–403
Writing changes to a database with COMMIT statement, 6–294 to 6–300