Oracle® Rdb for OpenVMS

New Features Manual

Release 7.4

May 2024

ORACLE®

Oracle Rdb New Features, Release 7.4.1.3 for OpenVMS

Primary Author: Rdb Engineering and Documentation group

Contents

Examples

Tables

Preface

Purpose of This Manual

This manual contains the New Features Chapters for Oracle Rdb Release 7.4.1.3 and prior Rdb 7.4 releases.

Deprecated and Desupported Features for Oracle Rdb

Each release of Oracle Rdb introduces behavior changes for your database in addition to new features. Changes in behavior include deprecated and desupported debug flags, parameters, options, syntax, and the deprecation and desupport of features and components.

Each chapter in this manual describes behavior changes where features have been deprecated or desupported in that release. By deprecate, we mean that the feature is no longer being enhanced but is still supported for the full life of the Oracle Rdb release. By desupported, we mean that Oracle will no longer fix bugs related to that feature and may remove the code altogether (see the *Obsolete Features* section in each chapter). Where indicated, a deprecated feature may be desupported in a future major release.

Access to Oracle Support

Oracle customers that have purchased support 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.

Document Structure

This manual consists of the following chapters:

Chapter 1	Describes enhancements introduced in Oracle Rdb Release 7.4.1.3
Chapter 2	Describes enhancements introduced in Oracle Rdb Release 7.4.1.2
Chapter 3	Describes enhancements introduced in Oracle Rdb Release 7.4.1.1
Chapter 4	Describes enhancements introduced in Oracle Rdb Release 7.4.1.0
Appendix A	Describes enhancements in the Optimizer
Appendix B	Describes RDO, RDBPRE and RDB$INTERPRET Features

1
Enhancements And Changes Provided in Oracle Rdb Release 7.4.1.3

1.1 Enhancements

1.1.1 The Lock_timeout Qualifier can now be Used With All RMU Reclaim Qualifiers

Previously, the RMU Reclaim Lock_timeout qualifier could only be specified if the RECLAIM Free_pages qualifier was also specified. With this release of Oracle Rdb the Lock_timeout qualifier can be specified for all RMU Reclaim command functions.

If another user is already accessing the current storage area or logical area to be processed the RMU Reclaim command will wait until the other user releases the storage area or logical area, or a lock wait timeout occurs. If no default lock timeout has been defined for the database or no current lock timeout is in effect, or the user wants to specify a lock timeout value to be used just during the execution of the current RMU Reclaim command, the Lock_timeout qualifier can be specified. If a lock wait timeout occurs, a **warning** message will be output that the storage area or logical area could not be processed because of a lock conflict with another user and a **warning** status will be returned by the RECLAIM command. That storage area or logical area will not be processed, but the RECLAIM command will continue processing the next storage area or logical area in the specified list of storage areas or logical areas.

To specify a temporary lock timeout value in seconds that will be in effect only during the execution of the current RMU Reclaim command the syntax is

```
/LOCK_TIMEOUT=seconds
```

The value that must be specified is the maximum time in seconds during which the current RMU Reclaim command will wait to acquire an EXCLUSIVE UPDATE lock on the current storage area or logical area to be processed when accessing an on-line database with other users.

If the Lock_timeout qualifier is not specified the lock wait time value used is the value of the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL (if defined), or the LOCK TIMEOUT INTERVAL specified by the SQL CREATE DATABASE or ALTER DATABASE statement. If neither value has been specified and the LOCK_TIMEOUT qualifier is not specified the RMU Reclaim command will wait indefinitely to acquire an EXCLUSIVE UPDATE lock on the current storage area or logical area to be processed.

If Lock_timeout=0 is specified, the RMU Reclaim command will ignore any lock timeout defaults that may be in effect and wait indefinitely to acquire an EXCLUSIVE UPDATE lock on the current storage area or logical area to be processed.

In the following example, warning messages are output, even if Log is not specified, in the RMU Reclaim commands if there is a lock conflict with another user that cannot be resolved. The optional Lock_timeout qualifier is specified to allow the user to specify a temporary lock timeout value to be used for the current RMU Reclaim command.

```
$ RMU/RECLAIM/NOLOG/AREA=AREA1/LOCK_TIMEOUT=600 SAMPLE.RDB
%RMU-W-RCLMARNOTPRC, Area AREA1 could not be processed due to a lock conflict
$
$ RMU/RECLAIM/LOG/LAREA=SAMPLE_TABLE/FREE_PAGES/LOCK_TIMEOUT=1200 -
$_  SAMPLE.RDB
%RMU-I-RCLMLAREA, Reclaiming logical area SAMPLE_TABLE in physical
    area DISK:[DIRECTORY]AREA1.RDA;1
%RMU-W-RCLMLARNOTPRC, Logical area SAMPLE_TABLE could not be processed
    due to a lock conflict
%RMU-I-RCLMLPAGPRC, 0 pages processed for logical area SAMPLE_TABLE in
    physical area DISK:[DIRECTORY]AREA1.RDA;1
%RMU-I-RCLMLPAGFREED, 0 clump pages freed for logical area
    SAMPLE_TABLE in physical area DISK:[DIRECTORY]AREA1.RDA;1
$
```

1.1.2 RMU Collect Optimizer_Statistics UNIQUE Index Performance Enhancement

Bug 3804717

A performance enhancement has been added to the RMU Collect Optimizer_ Statistics and RMU Show Optimizer_Statistics commands. If only cardinality is being collected and the index is unique with a single index segment, that index will not be analyzed to collect the index cardinality because the index cardinality is always derived from the base table. The index will be analyzed in all other cases.

If the Log qualifier is specified for the RMU Collect Optimizer_Statistics or Show Optimizer_Statistics commands a message will be output to indicate that the index did not have to be analyzed to collect index cardinality statistics.

```
$ RMU/COLLECT OPTIMIZER MF_PERSONNEL/STAT=CARDINALITY/LOG
Start loading tables... at 12-JAN-2024 10:44:50.49
Done loading tables.... at 12-JAN-2024 10:44:50.51
Start loading indexes... at 12-JAN-2024 10:44:50.51
%RMU-I-COLTXT_04, Unique Index COLL_COLLEGE_CODE for Table COLLEGES
    has same cardinality as table, cardinality not collected
%RMU-I-COLTXT_04, Unique Index DEPARTMENTS_INDEX for Table DEPARTMENTS
    has same cardinality as table, cardinality not collected
%RMU-I-COLTXT_04, Unique Index EMPLOYEES_HASH for Table EMPLOYEES has
    same cardinality as table, cardinality not collected
```

```
%RMU-I-COLTXT_04, Unique Index EMP_EMPLOYEE_ID for Table EMPLOYEES has
    same cardinality as table, cardinality not collected
Done loading indexes.... at 12-JAN-2024 10:44:50.56
Start collecting btree index stats... at 12-JAN-2024 10:44:50.76
Done collecting btree index stats.... at 12-JAN-2024 10:44:50.77
Start collecting table & hash index stats... at 12-JAN-2024 10:44:50.77
Done collecting table & hash index stats.... at 12-JAN-2024 10:44:50.79
Start calculating stats... at 12-JAN-2024 10:44:50.81
Done calculating stats.... at 12-JAN-2024 10:44:50.81
Start writing stats... at 12-JAN-2024 10:44:50.84
```

1.1.3 New Partitions Qualifier for RMU Unload Command

Enhancement 34940446

This release of Oracle Rdb supports a new Partitions qualifier as part of RMU Unload. The Partitions qualifier modifies the table being selected to restrict access to just those named partitions.

This change to RMU allows large partitioned tables to be unloaded by physical area, rather than selecting the whole table, or by unloading through a view that identifies selected rows.

The following example shows the names of the partitions, using SQL SHOW STORAGE MAP and several RMU Unload commands to unload each partition into a separate data file.

```
$ sql$
attach 'filename mf_personnel';
show storage map (partition) employees_map;
     EMPLOYEES_MAP
 For Table:       EMPLOYEES
 Placement Via Index:    EMPLOYEES_HASH
   .
   .
   .
 Partition information for storage map:
 Comment:     employees partitioned by "00200" "00400"
 Compression is:    ENABLED
  Partition: (1) SYS_P00079
   Storage Area: EMPIDS_LOW
  Partition: (2) SYS_P00080
   Storage Area: EMPIDS_MID
  Partition: (3) SYS_P00081
   Storage Area: EMPIDS_OVER
disconnect all;
$
$ rmu/unload mf_personnel employees/partition=(SYS_P00079) SYS_P00079.unl
%RMU-I-DATRECUNL,   37 data records unloaded 17-JAN-2024 09:42:13.45.
$ rmu/unload mf_personnel employees/partition=(SYS_P00080) SYS_P00080.unl
%RMU-I-DATRECUNL,   57 data records unloaded 17-JAN-2024 09:42:13.54.
$ rmu/unload mf_personnel employees/partition=(SYS_P00081) SYS_P00081.unl
%RMU-I-DATRECUNL,   6 data records unloaded 17-JAN-2024 09:42:13.64.
$
```

See also the Transaction_Type option PARTITION_LIST which can be used to reserve partitions for PROTECTED or EXCLUSIVE access to reduce locking during unload. See Section 1.1.4, New PARTITION_LIST Option for the Transaction_type Qualifier for more details.

Usage Notes

- One or more partition names specified as a comma separated list.

  ```
  $ rmu/unload -
      mf_personnel -
      employees/partitions=(SYS_P00079, SYS_P00081) -
      saved_rows.unl
  ```

  If Partitions=* is specified then this is synonymous with referencing the base table. If the Partitions qualifier lists all the partitions of the table then this is synonymous with Partitions=*.

- When Partitions is used the subset table is treated as READ ONLY and therefore Delete_rows is not permitted.

  ```
  $ rmu/unload -
      /delete_rows -
      mf_personnel -
      employees/partitions=(*) -
      saved_rows.unl
  %DCL-W-CONFLICT, illegal combination of command elements - check
  documentation
   \DELETE_ROWS\
  $
  ```

- When Partitions is used the indices on the table are not used by the unload query so the table will always be scanned using SEQUENTIAL ACCESS.

- The partition names are defined by the storage map for the table. When the storage map is created or altered a partition name may be provided for each storage area as part of the IN clause, or Rdb will generate a unique name using the logical area number assigned to each partition.

  The Oracle Rdb default name starts with SYS_P, such as SYS_P00079. Oracle recommends that more descriptive names be assigned to the storage map partitions. The existing names can be altered using the RENAME PARTITION clause of ALTER STORAGE MAP.

  ```
  SQL> alter storage map JOB_HISTORY_MAP
  cont>     rename partition SYS_P00089 to EMPIDS_LOW;
  SQL> alter storage map JOB_HISTORY_MAP
  cont>     rename partition SYS_P00090 to EMPIDS_MID;
  SQL> alter storage map JOB_HISTORY_MAP
  cont>     rename partition SYS_P00091 to EMPIDS_OVER;
  SQL> commit;
  ```

  Using the PARTITION clause when using the CREATE STORAGE MAP, and the ALTER STORAGE MAP statements allow the database administrator to provide meaningful names to simplify table management and partition references.

1.1.4 New PARTITION_LIST Option for the Transaction_type Qualifier

This release of Oracle Rdb includes the following new option on the RMU Unload Transaction_type qualifier.

- PARTITION_LIST=(partname1 [,...])

  This option accepts a list of partition names for the table being unloaded.

This option is compatible with the EXCLUSIVE, PROTECTED and SHARED options.

This example uses this definition of the EMPLOYEES_MAP

```
SQL> show storage map employees_map
     EMPLOYEES_MAP
 For Table:        EMPLOYEES
 Placement Via Index:    EMPLOYEES_HASH
 Partitioning is:    UPDATABLE
 Store clause:        Rename partition SYS_P00079 to EMPIDS_LOW
    Rename partition SYS_P00080 to EMPIDS_MID
    Rename partition SYS_P00081 to EMPIDS_OVER

 Partition information for storage map:
 Compression is:    ENABLED
  Partition: (1) EMPIDS_LOW
   Storage Area: EMPIDS_LOW
  Partition: (2) EMPIDS_MID
   Storage Area: EMPIDS_MID
  Partition: (3) EMPIDS_OVER
   Storage Area: EMPIDS_OVER
SQL>
```

The RMU Unload command in this example unloads a single partition EMPIDS_LOW using the Partitions qualifier and specifies EXCLUSIVE READ for that partition using the Transaction_type qualifier.

```
$ rmu/unload mf_personnel -
    /transaction=(exclusive, partition_list=empids_low) -
    employees/partitions=(empids_low) -
    empids_low.unl
%RMU-I-DATRECUNL,   37 data records unloaded ...
$
```

1.1.5 Oracle Rdb Compilers Now Set the Object Module Severity

This release of Oracle Rdb brings an enhancement to the SQL Module Language compiler and the SQL Precompiler. In line with other OpenVMS compilers these compilers from Oracle Corporation now also establish the severity in the generated object module. This is especially important when the compile results in generated SQL warning messages.

The following example shows that now such warnings also get detected by the LINK command.

```
$       sql$pre /cc /list-
            /object=TEST4.obj -
            TEST4.sc
    finish;        // disconnect default;
    1
%SQL-I-DEPR_FEATURE, (1) Deprecated Feature: FINISH is replaced by DISCONNECT
    insert into work_status (status_code) values (user);
                                                  1
%SQL-W-LENMISMAT, (1) Truncating right hand side string for assignment to column STATUS_CODE
$
$    link TEST4,sql$user/library
%LINK-W-WRNERS, compilation warnings
    in module MM file USERS_DISK:[TESTING]TEST4.OBJ;1
$
```

Oracle recommends that you correct the SQL sources that generate warnings, or disable the warnings using /NOWARNINGS for the SQL Module Language compiler and /SQLOPTIONS=NOWARN for the SQL Precompiler.

You can use the ANALYZE/OBJECT DCL command to examine this setting.

- On an Alpha system this attribute is shown in the END OF MODULE section.

  ```
  26.  END OF MODULE (EOBJ$C_EEOM), 10 bytes

      severity: warnings (1)
      Highest conditional linkage index: 40
  ```

- On an IA64 system it appears in the Elf Header Information.

  ```
  Elf Header Information, at file address 0.

      Class:                                      ELF-64
      Data:                                       Little-endian byte order
      Elf Header Version:                         1.
      OS ABI:                                     OpenVMS
      OS ABI Version:                             2.
      Type:                                       Object
      Machine Architecture:                       IA_64
      Elf File Version:                           1.
      VMS Completion Code:                        WARNING
  ```

1.1.6 New ALTER DATABASE ... REBUILD SYSTEM INDEX Clause

Bugs 5022392, 28854908, 35925682

This release of Oracle Rdb introduces a new REBUILD SYSTEM INDEX clause as part of ALTER DATABASE. This feature allows a database administrator to rebuild selected Rdb system table indices. Reasons to do this might be to improve retrieval due to restructuring or when instructed to do so by Oracle Support.

Syntax

```
REBUILD SYSTEM INDEX index-name
```

This example rebuilds the RDB$INTER_OBJ_SUBOBJ_NDX index that is defined upon the RDB$INTERRELATIONS table.

```
$ sql$
alter database
    filename MF_PERSONNEL
    rebuild system index RDB$INTER_OBJ_SUBOBJ_NDX
;
$
```

Usage Notes

- The ALTER DATABASE statement may only specify the REBUILD SYSTEM INDEX clause once. Use multiple ALTER DATABASE statements to rebuild multiple indices.
- The REBUILD SYSTEM INDEX clause requires exclusive access to the system metadata and is therefore an OFFLINE clause. The change is implicitly committed.
- Because ALTER DATABASE can not be rolled back Oracle recommends that a full RMU BACKUP be taken before executing this command.
- The REBUILD SYSTEM INDEX clause is incompatible with ADD CACHE, ADD JOURNAL, ADD STORAGE AREA, ALTER CACHE, ALTER JOURNAL, ALTER STORAGE AREA, DROP CACHE, DROP JOURNAL, DROP STORAGE AREA, and RESERVE clauses. Specify these clauses in a separate ALTER DATABASE statement.
- The index name provided to the REBUILD SYSTEM INDEX clause must be one of the defined system table indices. Use the SQL SHOW SYSTEM TABLES statement to see the name of the system tables.

```
SQL> show system tables
System tables in database with filename SQL$DATABASE
     RDB$COLLATIONS
     RDB$CONSTRAINTS
     RDB$CONSTRAINT_RELATIONS
     RDB$DATABASE
     RDB$FIELDS
     RDB$FIELD_VERSIONS
     RDB$GRANTED_PROFILES
     RDB$INDEX_SEGMENTS
     RDB$INDICES
     RDB$INTERRELATIONS
   .
   .
   .
SQL>
```

  Use the SQL SHOW TABLE (INDEX) statement to display the names of the system indices for a table.

```
SQL> show table (index) RDB$INTERRELATIONS
Information for table RDB$INTERRELATIONS

Compression is disabled.

Indexes on table RDB$INTERRELATIONS:
RDB$INTER_CON_NAM_NDX           with column RDB$CONSTRAINT_NAME
  Duplicates are allowed
  Type is Sorted Ranked
    Duplicates are Compressed Bitmaps
  Key suffix compression is DISABLED

RDB$INTER_ENTITY_NDX            with column RDB$ENTITY_NAME1
                                and column RDB$ENTITY_NAME2
  Duplicates are allowed
  Type is Sorted Ranked
    Duplicates are Compressed Bitmaps
  Key suffix compression is DISABLED
```

```
RDB$INTER_OBJ_SUBOBJ_NDX        with column RDB$OBJECT_NAME
                                and column RDB$SUBOBJECT_NAME
  Duplicates are allowed
  Type is Sorted Ranked
    Duplicates are Compressed Bitmaps
  Key suffix compression is DISABLED
```

- Oracle recommends using the RDMS$SET_FLAGS logical name defined to the value "INDEX_STATS" prior to running interactive SQL. This will enable tracing during execution of the rebuild.

```
$ define/user RDMS$SET_FLAGS "INDEX_STATS"
$ sql$
alter database
    filename MF_PERSONNEL
    rebuild system index RDB$INTER_OBJ_SUBOBJ_NDX
;
   .
   .
   .
$
```

- A small subset of the Rdb system indices are required during the rebuild action and may not be rebuilt using the REBUILD SYSTEM INDEX clause.

1.1.7 New Implementation of the SQL IN Predicate

In prior releases of Oracle Rdb SQL would expand the IN predicate into a series of equivalent OR and equal (=) conditions. In this release of Oracle Rdb a new compact format is used to encode and transmit queries from the client to the server. This format significantly reduces the size of the queries, triggers, constraints and procedure definitions that use the IN predicate, with related reductions in processing times for the query compiler. This new format also allows several opportunities for new and future optimizations. See *Section A.1.7, New Optimizations for the SQL IN Predicate* for more details and examples of the optimizations available in this release of Oracle Rdb.

Usage Notes

- The generated query has a different structure from that generated in prior versions. Any query outlines that are activated by matching the query id value will no longer match. Customers are advised to recreate the query outlines to match the new query id. If applications use the OPTIMIZE USING clause then that name will still be used to locate the appropriate outline and no changes should be required.
- The current releases of Oracle CDD Repository, Replication Option for Rdb, and Oracle TRACE do not support this new format. This support is forthcoming for these products (V7.4.1). Until such updated versions are installed Oracle recommends that customers define this logical name system wide.

```
$ define /system SQL$DISABLE_IN_CLAUSE "Y"
```

Use RMU/Show Logical_name SQL$DISABLE_IN_CLAUSE /Description to read more information.

- Applications that query via the remote interface to older Oracle Rdb releases (7.4.1.2 and earlier) will adjust to the capabilities of that version when using Interactive SQL, or Dynamic SQL.

  However, if the SQL Module Language compiler or the SQL Precompiler are used to compile applications using the current version of Oracle Rdb but run against a different run-time database on an older Rdb version then it is possible that the IN predicate will fail to execute. Define this logical name prior to compiling any source modules.

```
$! TEST A: go remote to see that the support fails
$! >>> expect: %RDB-E-INVALID_BLR, request BLR is incorrect at offset ...
$ SQL$PRE/CC-
     /OBJECT=SCC_NEW_IN_CLAUSE -
     SCC_NEW_IN_CLAUSE
$ LINK SCC_NEW_IN_CLAUSE,SQL$USER/LIBRARY
$
$ define/user TARGET_DATABASE SCC_NEW_IN_CLAUSE_DB_REM
$ RUN SCC_NEW_IN_CLAUSE
%RDB-E-INVALID_BLR, request BLR is incorrect at offset 226
$
$! TEST B: go remote to see that it now work after compiling with disabled clause
$ define/user SQL$DISABLE_IN_CLAUSE "Y"
$ SQL$PRE/CC-
     /OBJECT=SCC_NEW_IN_CLAUSE -
     SCC_NEW_IN_CLAUSE
$ LINK SCC_NEW_IN_CLAUSE,SQL$USER/LIBRARY
$
$ define/user TARGET_DATABASE SCC_NEW_IN_CLAUSE_DB_REM
$ RUN SCC_NEW_IN_CLAUSE
last-name -> Canonica
last-name -> Dietrich
last-name -> Gehr
last-name -> Johnson
last-name -> O'Sullivan
last-name -> Tarbassian
last-name -> Wood
last-name -> Ziemke
8 rows fetched.
$
```

  Applications compiled with prior versions of Oracle Rdb (7.4.1.2 or earlier) will still use the older query format. Such applications only need to be recompiled to take advantage of the new optimizations.

1.1.8 Updated Sample Code for RDB_CYPHER

This release of Oracle Rdb includes a new sample program RDB_CYPHER. It is written in the C language and uses a different approach to using Rdb modules to provide context for many calls to the OpenVMS Encryption services.

Note

This program supersedes the original sample program RDB_CYPHER.B32. That older example source code is still provided as reference.

Introduction

This sample program demonstrates calling encryption and description routines provided by the OpenVMS ENCRYPT library, but could also be adapted to locally written, or open source routines.

Note

Please refer to the *VSI OpenVMS Utility Routines Manual, Chapter 11 Encryption (ENCRYPT) Routines* for more details on the routines referenced in the sample SQL external function definitions.

The Encryption Routines call interface is also described by DCL HELP ENCRYPT_Routine.

The sample program is described in these sections:

1. Choosing the interface.
2. Implementing the external module and routines.
3. Compiling, linking and deploying this module.
4. Important notes for the implementation.

Oracle recommends a study of the encryption services be made before reading further.

Choosing the Interface

OpenVMS provides two routines to encrypt and decrypt a single field; ENCRYPT$ENCRYPT_ONE_RECORD and ENCRYPT$DECRYPT_ONE_RECORD. These routines are called to encrypt each table column values, and subsequently decrypt those column values for display or other computation.

Two external routines are defined to accept input and call the appropriate encryption service. Note that these services require the use of a defined key which in this sample code is created by the routine ENCRYPT$DEFINE_KEY but could also be defined outside the application using DCL commands; ENCRYPT /CREATE_KEY.

For the purposes of this example the routines ENCRYPT$DEFINE_KEY and ENCRYPT$DELETE_KEY are called to manage the key.

Implementing the External Module and Routines

Applications that include encryption support are expected to call these routines frequently. Therefore, the calls to manage the key are not part of the per column encryption and decryption processing since they need not be repeated. Instead they are implemented in the external module's NOTIFY routine.

Each external routine definition can reference a routine (that can not be written in SQL) which is called at special intervals during external routine usage. From our example when the database connect (attach) occurs the first activation of routines in the module will execute the CONNECT action (RDB_K_RTX_

NOTIFY_CONN_START[1]). This is specified using the NOTIFY clause as part of the EXTERNAL routine definition.

```
notify ENCD_NOTIFY on connect
```

Although Rdb supports standalone external routines for the purposes of this example we need a common context. Therefore, all the external routines are defined within a single module (see the CREATE MODULE Statement in the Oracle Rdb SQL Reference Manual).

The support code for the external routines will be written in C, and the module will include three routines: ENCD_NOTIFY; the notify routine, ENCRYPT_CCN; routine to encrypt the credit card number, and DECRYPT_CCN; routine to decrypt the credit card number.

Any database intending to call ENCRYPT_CCN and DECRYPT_CCN must have their interfaces defined in the database so that queries can reference them by name and so the parameter can be passed correctly. These SQL CREATE statements are included in the SQL$SAMPLE:RDB_CYPHER.C source code. Extract them and create a SQL command procedure which can be used to define the interface on any databases that need these services.

```
create module ENCRYPTION_SERVICES
  procedure ENCRYPT_CCN (
      in :plain_text_dsc        char (20) character set unspecified by descriptor,
      out :cypher_text_dsc      char (24) character set unspecified by descriptor);
     external
      name ENCRYPT_CCN
      location 'SYS$SHARE:RDB_CYPHER.EXE' with all logical_name translation
      language C
      parameter style GENERAL
      notify ENCD_NOTIFY on connect
      comment is 'Encrypt the CCN using a predefined key'
      ;
  procedure DECRYPT_CCN (
      in :cypher_text_dsc       char (24) character set unspecified by descriptor,
      out :plain_text_dsc       char (20) character set unspecified by descriptor);
     external
      name DECRYPT_CCN
      location 'SYS$SHARE:RDB_CYPHER.EXE' with all logical_name translation
      language C
      parameter style GENERAL
      notify ENCD_NOTIFY on connect
      comment is 'Decrypt the CCN using a predefined key'
      ;
end module;
```

Once created the module should be granted (using the SQL GRANT statement) appropriate protections on the module as required by the security policies of the organization. Note that individual routines in a module do not have separate protections.

[1] See SYS$LIBRARY:DSRI.H

Compiling, Linking and Deploying this Module

The sample module must be compiled and linked. The resulting executable image should then be copied to SYS$SHARE. The image should also be installed to ensure correct and valid usage.

```
$ CC RDB_CYPHER.C
$ LINK /SHARE RDB_CYPHER.OBJ+SYS$INPUT:/OPTION
symbol_vector=( -
    ENCRYPT_CCN=procedure, -
    DECRYPT_CCN=procedure, -
    ENCD_NOTIFY=procedure)
$ COPY RDB_CYPHER.EXE SYS$COMMON:[SYSLIB] /PROT=W:RE
$ INSTALL REPLACE SYS$SHARE:RDB_CYPHER.EXE /OPEN/SHARE/HEAD
```

Now when SQL queries, triggers or stored procedures execute the ENCRYPT_CCN and DECRYPT_CCN routines the shareable image will be activated from a known secure location.

Important Notes for the Implementation

This C module is supplied solely as a template example of calling the OpenVMS encryption routines from an Rdb stored procedure. The expected use of this module would be to encrypt and decrypt a 20 byte text field containing a credit card number.

You are free to use this example code and extend it; adding more routines for different sized fields, changing the key definition module, and so on.

Further considerations:

- How is NULL handled? External routines do not accept NULL values so the interface might need a jacket SQL routine that passes the nullness of the data as a second parameter.
- If columns include duplicate data for the same table it might be more secure to add salting (randomized values) to the plain text prior to encryption to obscure any patterns. Upon decrypting that randomized data is not returned to the caller.
- This example only uses a single key for inception but different columns could have different encryption keys. That may allow, for instance, different users to decrypt a subset of the data based on the application requirements.
- Protect the KEY. If the key is lost then the encrypted data is no longer accessible, so don't loose it. Also protect the security of the encryption key. Coding it as plain text in this source file is for example purposes only.

1.2 Obsolete Features

None for this release.

2 Enhancements And Changes Provided in Oracle Rdb Release 7.4.1.2

2.1 Enhancements

2.1.1 Substring Function Changes

This release of Oracle Rdb provides native versions of the functions SUBSTR and SUBSTRB which were previously provided as external functions in the SQL_FUNCTIONSnn.SQL script.

Note

Those external functions will remain defined in the SQL_FUNCTIONS74.SQL function script to support applications that currently use them, or wish to continue to use those functions.

These new native functions have the following advantages over the external functions:

- These functions are now part of the Oracle Rdb SQL language and no external functions for SUBSTR or SUBSTRB need be defined in the database.
- These functions are polymorphic; meaning they adapt to the passed character string - both length and character set. Therefore, the result will be derived from the length of the input string length and the length field (if specified).

  The SQL external functions defined in SQL_FUNCTIONS74.SQL were typically limited to CHAR and VARCHAR inputs of 2000 characters, and returned a VARCHAR (2000) typed result even when operating on shorter string inputs.
- These builtin functions now accept CHAR and VARCHAR strings up to 65535 octets in length (the actual length in CHARACTERS will depend on the character set being used).
- These changes will typically result in much lower virtual memory usage and better run-time performance for queries. In particular applications using ORACLE dialects (such as those using OCI Services for Rdb) should expect a considerable reduction in query size.

Note

Applications created using the SQL Module Language or SQL Pre-compiler in prior releases will continue to use the SQL external functions until those application sources are recompiled.

The substring family of functions operate similarly with these differences:

- SUBSTRING (char_string FROM start_position [FOR substring_length])

  This is the ANSI and ISO SQL Database Language implementation. Optional USING CHARACTERS or USING OCTETS clauses adjusts the interpretation of the start_position and substring_length.

- SUBSTR (char_string , start_position [, substring_length])

  For SUBSTR start_position and substring_length represent units as characters. SUBSTR is similar in operation to SUBSTRING with the USING CHARACTERS clause with these extra rules.

  * If start_position is 0, then it is treated as 1.
  * If start_position is positive, then Oracle Rdb counts from the beginning of char_string to find the first character of the result.
  * If start_position is negative, then Oracle Rdb counts backward from the end of char_string. For example, to return the last 3 characters of a string you can pass start_position as -3 and the substring_length as 3. If the computed beginning of char_string is prior to the start of the string then a zero length string is the result.

- SUBSTRB (char_string , start_position [, substring_length])

  For SUBSTRB start_position and substring_length represent units as octets. SUBSTRB is similar in operation to SUBSTRING with the USING OCTETS clause with the same rules as described for SUBSTR.

If the dialect is set to an Oracle dialect (ORACLE LEVEL1, ORACLE LEVEL2, or ORACLE LEVEL3) these functions will return NULL if the resulting string was zero length.

Changes in Query Signature related to Query Outlines

The database administrator should be aware that queries using SUBSTR or SUBSTRB generated by SQL will be differently structured from queries using the SQL external functions even if the results remain the same.

Similarly, the SUBSTRING builtin function now directly supports the Oracle dialect semantics which consider a zero length string result as being equivalent to NULL. So when using SUBSTRING under the Oracle dialects (ORACLE LEVEL1, ORACLE LEVEL2, and ORACLE LEVEL3) the actual query will be changed.

Therefore, any query outline previously created for such queries will need to be redefined. The generated query id (signature) changes even if the strategy does not.

Consider this example which shows that under Oracle Rdb V7.3 SUBSTRING queries for Oracle Dialects were rewritten with CASE expressions to ensure the correct result semantics.

```
SQL> select employee_id, last_name, first_name
cont> from employees
cont> where substring (last_name from 2 for 1) = ''''
cont> ;
~S: Outline "QO_0_73_MODE_3" used
Tables:
  0 = EMPLOYEES
Leaf#01 FFirst 0:EMPLOYEES Card=100
  Bool: CASE (WHEN (CHAR_LENGTH (SUBSTRING (0.LAST_NAME FROM 2 FOR 1)) = 0)
         THEN NULL ELSE SUBSTRING (0.LAST_NAME FROM 2 FOR 1)) = '''
  BgrNdx1 EMP_LAST_NAME [0:0] Fan=12
    Bool: CASE (WHEN (CHAR_LENGTH (SUBSTRING (0.LAST_NAME FROM 2 FOR 1)) = 0)
           THEN NULL ELSE SUBSTRING (0.LAST_NAME FROM 2 FOR 1)) = '''
 EMPLOYEE_ID   LAST_NAME        FIRST_NAME
 00171         D'Amico          Aruwa
 00190         O'Sullivan       Rick
2 rows selected
SQL>
```

In this case a query outline QO_0_73_MODE_3 was defined (and was used) which matched the query signature. However, when run under this release of Oracle Rdb with a simplified query the outline is no longer matched.

```
SQL> select employee_id, last_name, first_name
cont> from employees
cont> where substring (last_name from 2 for 1) = ''''
cont> ;
Tables:
  0 = EMPLOYEES
Leaf#01 FFirst 0:EMPLOYEES Card=101
  Bool: SUBSTRING (0.LAST_NAME FROM 2 FOR 1) = '''
  BgrNdx1 EMP_LAST_NAME [0:0] Fan=12
    Bool: SUBSTRING (0.LAST_NAME FROM 2 FOR 1) = '''
 EMPLOYEE_ID   LAST_NAME        FIRST_NAME
 00171         D'Amico          Aruwa
 00190         O'Sullivan       Rick
2 rows selected
SQL>
```

Either the query could be modified to explicitly use the OPTIMIZE USING clause or a new query outline created to match the new query signature.

2.1.2 Support for EMPTY_BLOB and EMPTY_CLOB in SQL

This release of Oracle Rdb adds the new SQL functions EMPTY_BLOB () and EMPTY_CLOB ().

Purpose

EMPTY_BLOB and EMPTY_CLOB return an empty handle that can be used to initialize a LIST OF BYTE VARYING variable or as a value expression in an INSERT, UPDATE, or REPLACE statement to initialize a LIST OF BYTE VARYING column. An EMPTY indication means that the LIST is initialized, but not populated with data.

Examples

The following example initializes the BADGE_PHOTO column of the sample EMPLOYEES table to EMPTY using the UPDATE statement.

```
SQL> alter table EMPLOYEES
cont>     add column BADGE_PHOTO list of byte varying
cont> ;
SQL>
SQL> update EMPLOYEES
cont>     set BADGE_PHOTO = empty_blob ();
101 rows updated
SQL>
```

The following example initializes the BADGE_PHOTO column of the sample EMPLOYEES table to EMPTY using the INSERT statement.

```
SQL> insert into EMPLOYEES
cont>     values ('99001', 'McAlister', 'James', 'A',
cont>             '33 Running Deer Road', '', 'Hollis', 'NH', '03049',
cont>             'M', date vms'3-May-1989', '2',     /* part-time */
cont>             empty_blob()                        /* no photo yet */
cont>             );
1 row inserted
SQL>
```

2.1.3 New ANY_VALUE Aggregate Function

This release of Oracle Rdb adds the SQL function ANY_VALUE.

Format

```
ANY_VALUE ( [ ALL | DISTINCT ] value_expr ) [ FILTER ( WHERE boolean_expr ) ]
```

Arguments

- ALL, and DISTINCT: These keywords are supported by ANY_VALUE although they have no effect on the result of the query and are effectively ignored.
- value_expr: The expression can be a column, constant, variable, function or a complex expression.
- boolean_expr: the FILTER clause can be used to further filter the rows that participate in the aggregation.

Description

ANY_VALUE is an aggregate function that returns a single non-deterministic value of value_expr.

Use ANY_VALUE to optimize a query that has a GROUP BY clause. ANY_ VALUE returns a value of an expression in a group. It is optimized to return the first value.

It ensures that there are no comparisons for any incoming row and also eliminates the necessity to specify every column as part of the GROUP BY clause. Because it does not compare values, ANY_VALUE returns a value more quickly than MIN or MAX in a GROUP BY query.

NULL values in the expression are ignored.

Returns any value within each group based on the ORDER BY specification. Returns NULL if all rows in the group have NULL expression values.

Example

The following example shows the use of ANY_VALUE. When computing the aggregation of the rows values of SALARY_END which are NULL are excluded. In cases where all rows are excluded the ANY_VALUE function returns NULL.

```
SQL> select
cont>     employee_id, any_value (salary_end), count(*)
cont>     from salary_history
cont>     where employee_id in ('00241', '00227', '00225', '00164')
cont>     group by employee_id
cont> ;
 EMPLOYEE_ID
 00164         2-Mar-1981                    4
 00225        NULL                           1
 00227        NULL                           1
 00241        NULL                           1
4 rows selected
SQL>
```

2.1.4 Updated Diagnostics for the RMU REPAIR Command

With this release of Oracle Rdb the RMU Repair command has been updated to output improved messages when changes are made to the database.

- %RMU-W-FULLAIJBKUP, partially-journaled changes made; database may not be recoverable

  This message warns the user that, even though After Image Journaling is enabled, not all structural changes made to the database by the Repair operation are journaled. Therefore, the user should consider backing up the database once the repair operation completes.

  This message is only output if After Image Journaling is enabled for the database.

  ```
  $ RMU/REPAIR/ABM MF_PERSONNEL
  %RMU-W-FULLAIJBKUP, partially-journaled changes made; database may not
   be recoverable
  %RMU-W-DOFULLBCK, full database backup should be done to ensure future recovery
  ```

- %RMU-W-DOFULLBCK, full database backup should be done to ensure future recovery

  This message warns the user to do a full database backup after the repair operation completes to ensure that the structural changes made by the Repair operation can be restored by the RMU Restore command and subsequent RMU Recover from database AIJ files.

- %RMU-I-REPAIR_DEF, operation not specified; Reconstructing the SPAM pages for specified areas

This message informs the user that with no directives included on the RMU Repair command it defaulted to the /SPAM operation.

```
$ RMU/REPAIR MF_PERSONNEL
%RMU-I-REPAIR_DEF, operation not specified; Reconstructing the SPAM
 pages for specified areas
%RMU-W-DOFULLBCK, full database backup should be done to ensure future
 recovery
```

2.1.5 Updated Diagnostics for the RMU MOVE_AREA Command

With this release of Oracle Rdb the RMU Move_Area command has been updated to output improved messages when changes are made to the database.

- %RMU-W-FULLAIJBKUP, partially-journaled changes made; database may not be recoverable

  This message warns the user that the actual move of database files to another device or directory is not journaled. Therefore, the user should consider performing a full backup of the database once the move operation completes.

  The RMU Restore of the full database backup will recover the device and directory changes made and a subsequent RMU Recover will update the moved areas as needed.

  The following example shows the %RMU-W-FULLAIJBKUP message being output for a move operation which moves the database root file of a database with After Image Journaling enabled.

  ```
  $ RMU/MOVE_AREA /ROOT=disk:[directory] /NOLOG MF_PERSONNEL
  %RMU-W-FULLAIJBKUP, partially-journaled changes made; database may not
      be recoverable
  %RMU-I-AIJRSTAVL, 1 after-image journal available for use
  %RMU-I-AIJRSTMOD, 1 after-image journal marked as "modified"
  %RMU-I-AIJISON, after-image journaling has been enabled
  %RMU-W-DOFULLBCK, full database backup should be done to ensure future
      recovery
  ```

2.1.6 Hash Join Enhancements and Support

Status: BETA

Introduction

Table joins are typically solved using a mix of index and sorting strategies. In some cases there may not be any suitable index to use resulting in the solution selected by the optimizer requiring an extra sorting step.

The use of Hash Join eliminates the need to sort data by using an in-memory hash table to filter the join data.

This feature can be enabled by using any one of the following methods:

- Using the OPTIMIZE FOR HASH JOIN clause

  The optimizer will attempt to apply Hash Join on joins within the query. See SQL HELP SELECT for more information.

- Using SET FLAGS 'HASHING(JOINS)' statement

The optimizer will attempt to apply Hash Join on joins within all subsequent queries within this session. Use SET FLAGS 'NOHASHING' to disable this action within the session.

- Define the logical name RDMS$SET_FLAGS "HASHING(JOINS)"

  Similar to using the SET FLAGS statement, defining RDMS$SET_FLAGS to the value "HASHING(JOINS)" will enable this feature and will be applied to any applications that perform joins. Deassign this logical name (as well as RDMS$ENABLE_HASH_JOIN, see below) to disable this action within the process.

- Define the logical name RDMS$ENABLE_HASH_JOIN to true

  Defining this logical name to true ("T", "t", "Y", "y" or "1") instructs the Rdb optimizer to attempt to use in-memory Hash Join to solve queries. Deassign this logical name (as well as RDMS$SET_FLAGS, see above) to disable this action within the process.

  Use the RMU /SHOW LOGICAL RDMS$ENABLE_HASH_JOIN /DESCRIPTION command to see details.

- Create a query outline

  The outline should use the JOIN BY HASH clause to adjust specific parts of the query solution. Refer to SQL HELP CREATE OUTLINE for further information on query outlines.

  The created query outline will be used when referenced by name in the OPTIMIZE USING clause, or by any query matching the query signature (query outline id).

Oracle Corporation appreciates any and all feedback on this feature.

Changes for This Release

This release of Oracle Rdb includes an improved design of the Hash Join feature with these enhancements:

- Reduced VM usage for Hash Join solution

  Copies of the join keys is reduced for duplicate values.

- Reduced CPU usage during Hash Join operation

  Previously CPU usage increased as more duplicates were encountered in the hash table entries. The current version reduces the CPU time by no longer re-scanning the duplicate chains.

- Relaxed restrictions when Hash Join is using sequential retrieval

  Sequential retrieval is now more widely applied when no indices are defined or available for the join columns.

- Improved FULL OUTER JOIN support

  Wrong results related to Hash Join strategy, that occurred with Full Outer Join in prior releases, have been corrected.

- Improved diagnostics reported when Hash Join is rejected

  The SET FLAGS statement or the equivalent logical name RDMS$SET_FLAGS definition can be used to enable the display of these diagnostics. See the Examples.

Current Restrictions

Currently, the use of Hash Join may be rejected by the optimizer for a number of reasons as specified below. The reported reasons for the rejection can be displayed by running the query after setting the flags STRATEGY and DETAIL(2).

1. CHAR or VARCHAR characteristics of Index Columns differ

   When CHAR or VARCHAR columns are used during hashing the length, character set and collating sequence of the compared columns must be the same.

   ```
   Example:
   ~Sh: Hash Join #!2S inner rejected-TAB1_NDX and (generated-index)
       column 0 lengths differ
   ~Sh: Hash Join #!11S inner rejected-TAB1_NDX and (generated-index)
       column 1 collating sequences differ
   ~Sh: Hash Join #!3S inner rejected-TAB1_NDX and (generated-index)
       column 2 character sets differ
   ```

2. Data types in index columns differ

   The columns used for hashing must have the same datatypes.

   ```
   Example:
   ~Sh: Hash Join #5S inner rejected-TAB_NDX and (generated-index)
       column 0 datatypes differ
   ```

3. Date/time types in index columns differ

   If the join is between TIMESTAMP and DATE VMS columns then Oracle Rdb will consider this as non-matching data type.

   ```
   Example:
   ~Sh: Hash Join #!ULS inner rejected-TAB3_NDX and (generated-index)
       column 2 date/time types differ
   ```

4. The number of index columns differ

   The number of columns to compare in both streams must be the same.

   ```
   Example:
   ~Sh: Hash Join #6S inner rejected-TAB1_NDX and (generated-index)
       number of columns differ
   ```

5. Sort orders in index columns differ

   The sort ordering for columns (ASC and DESC clauses) must match.

   ```
   Example:
   ~Sh: Hash Join #!ULS inner rejected-TAB2_NDX2 and (generated-index)
       column 1 sort orders differ
   ```

6. Integer precision in index columns differ

   Hash Join cannot be used if it is required to compare integer columns with different precision.

   ```
   Example:
   ~Sh: Hash Join  #6S inner rejected-TAB1_NDX and TAB2_NDX column 0 scales differ
   ```

7. OR Retrieval on outer or inner table

   Hash Join cannot be used if either stream relies on the results of an OR-retrieval optimization.

   ```
   Example:
   ~Sh: Hash Join #9S inner rejected-OR retrieval on inner table
   ~Sh: Hash Join #19S outer rejected-OR retrieval on outer table
   ```

8. Outer or inner column not equivalent with join columns

   Hash Join can only be done using columns that are considered to be in the same equivalence group and thus may be compared.

   ```
   Example:
   ~Sh: Hash Join #12S outer rejected-X1 column 0 not equivalent with join columns
   ~Sh: Hash Join #3S inner rejected-(generated-index)
       column 0 not equivalent with join columns
   ```

9. Index columns not provided as join columns

   Hash join using the named index will be rejected if not all index columns are provided values (no partial index key matches are allowed).

   ```
   Example:
   ~Sh: Hash Join #1S outer rejected-values of all outer
       TAB1_NDX columns not provided
   ~Sh: Hash Join #16S inner rejected-values of all inner
       TAB2_NDX columns not provided
   ```

10. Outer or inner index column not a join predicate

    Hash Join can only be done when the matching columns for the inner and outer streams take part in the join predicate.

    ```
    Example:
    ~Sh: Hash Join #15S inner rejected-TAB1_NDX column 1 not a join predicate
    ```

11. Outer or inner stream not a table

    Hash Join requires that streams access a table directly or via an index.

    ```
    Example:
    ~Sh: Hash Join #11S inner rejected-inner stream not a table
    ~Sh: Hash Join #14S outer rejected-outer stream not a table
    ```

12. Outer or inner stream not comply with outlines

    Hash Join cannot be done if a stream cannot be optimized in accordance with the supplied outline.

    ```
    Example:
    ~Sh: Hash Join #4S inner rejected-does not comply with outline
    ~Sh: Hash Join #12S outer rejected-does not comply with outline
    ```

13. Query contains aggregate outer join boolean

    Hash Join is not available if the outer join relies on a check on the results of an aggregation.

    ```
    Example:
    ~Sh: Hash Join #3S inner rejected-Q2 contains Aggregate OJ boolean
    ```

14. Reverse scan on outer or inner table

    If either stream uses Reverse Scan then Hash Join is not possible.

    ```
    Example:
    ~Sh: Hash Join #7S outer rejected-reverse scan on outer table
    ```

15. Sequential retrieval on outer or inner table not available

    If the value to be matched is not derived from an actual or derived column Hash Join cannot utilize Sequential Retrieval

    ```
    Example:
    ~Sh: Hash Join #4S outer rejected-sequential retrieval on outer table
    ~Sh: Hash Join #12S inner rejected-sequential retrieval on inner table
    ```

Examples

This section contains examples of attempts to use Hash Join when restrictions are encountered.

The following query reverts to cross strategy when full compliance with the outline was not possible:

First, create a query outline. This example uses an Rdb generated outline for the query produced using SET FLAGS 'OUTLINE' with the name LOJ_NEST_HSH.

Example 2–1 Example 1: Using OPTIMIZE USING

```
drop outline LOJ_NEST_HSH;
create outline LOJ_NEST_HSH
id 'FE43684748E33D965A065EF6545EB1BB'
mode 0
as (
  query (
-- For loop
    subquery (
      subquery (
        P 2     access path sequential
        )
        join by cross to
      subquery (
        S 0     access path sequential
          join by hash to
        P 1     access path sequential
        )
      )
    )
  )
compliance optional     ;
```

Now execute the query with this query outline.

```
set flags 'strategy,detail(2)';
```

```
select snum, s.city, status, pnum, p.city, weight
  from s left join p
    on s.city = p.city and
       (exists (select pnum from p where pnum = 'P0'))
OPTIMIZE USING LOJ_NEST_HSH
;
~S: Outline "LOJ_NEST_HSH" used
~Sh: Hash Join #3S inner rejected-Q2 contains Aggregate OJ boolean
~S: Full compliance with the outline was not possible
Tables:
  0 = S
  1 = P
  2 = P
```

(continued on next page)

Example 2–1 (Cont.) Example 1: Using OPTIMIZE USING

```
Cross block of 2 entries  Q1
  Cross block entry 1
    Aggregate-F1: 0:COUNT-ANY (<subselect>) Q3
    Conjunct: 2.PNUM = 'P0'
    Get     Retrieval sequentially of relation 2:P  Card=6  JCard=1
  Cross block entry 2
    Match    (Left Outer Join)  Inner_TTBL Q2
      Outer loop
      Match_Key:0.CITY
        Sort: 0.CITY(a)
        Get Retrieval sequentially of relation 0:S  Card=5  JCard=5
      Inner loop
      Match_Key:1.CITY
        Temporary relation
        Sort: 1.CITY(a)
        Get Retrieval sequentially of relation 1:P  Card=6  JCard=10
S.SNUM   S.CITY           S.STATUS   P.PNUM   P.CITY            P.WEIGHT
 S5       Athens                30   NULL     NULL                  NULL
 S1       London                20   NULL     NULL                  NULL
 S4       London                20   NULL     NULL                  NULL
 S2       Paris                 10   NULL     NULL                  NULL
 S3       Paris                 30   NULL     NULL                  NULL
5 rows selected
```

If the query outline definition is changed to use COMPLIANCE MANDATORY then the query using it will fail to run if JOIN BY HASH is not possible.

```
create outline LOJ_NEST_HSH
id 'FE43684748E33D965A065EF6545EB1BB'
mode 0
as (
  query (
-- For loop
    subquery (
      subquery (
        P 2     access path sequential
        )
        join by cross to
      subquery (
        S 0     access path sequential
          join by hash to
        P 1     access path sequential
        )
      )
    )
  )
compliance mandatory ;
```

(continued on next page)

Example 2–1 (Cont.) Example 1: Using OPTIMIZE USING

```
set flags 'strategy,detail(2)';

select snum, s.city, status, pnum, p.city, weight
  from s left join p
    on s.city = p.city and
       (exists (select pnum from p where pnum = 'P0'))
OPTIMIZE USING LOJ_NEST_HSH
;
~S: Outline "LOJ_NEST_HSH" used
~Sh: Hash Join #3S inner rejected-Q2 contains Aggregate OJ boolean
~S: Full compliance with the outline was not possible
%RDB-E-QRYCOMP_FAILED, query or routine failed to compile
-RDMS-F-OUTLINE_FAILED, could not comply with mandatory query outline directives
```

The following query reverts to cross strategy when full compliance with the query outline is not possible.

Example 2–2 Example 2: Using the OPTIMIZE FOR HASH JOIN clause

```
select snum, s.city, status, pnum, p.city, weight
  from s left join p
    on s.city = p.city and
       (exists (select pnum from p where pnum = 'P0'))
OPTIMIZE FOR HASH JOIN
;
~S: Outline "QO_FE43684748E33D96_00000000" used
~Sh: Hash Join abandoned because Q2 contains Aggregate OJ boolean
~S: Full compliance with the outline was not possible
Tables:
  0 = S
  1 = P
  2 = P
Cross block of 2 entries  Q1
  Cross block entry 1
    Aggregate-F1: 0:COUNT-ANY (<subselect>) Q3
    Conjunct: 2.PNUM = 'P0'
    Get     Retrieval sequentially of relation 2:P
  Cross block entry 2
    Match    (Left Outer Join)  Inner_TTBL Q2
      Outer loop
      Match_Key:0.CITY
        Sort: 0.CITY(a)
        Get Retrieval sequentially of relation 0:S
      Inner loop
      Match_Key:1.CITY
        Temporary relation
        Sort: 1.CITY(a)
        Get Retrieval sequentially of relation 1:P
```

(continued on next page)

Example 2–2 (Cont.) Example 2: Using the OPTIMIZE FOR HASH JOIN clause

```
 S.SNUM   S.CITY           S.STATUS   P.PNUM   P.CITY           P.WEIGHT
 S5       Athens                 30   NULL     NULL                 NULL
 S1       London                 20   NULL     NULL                 NULL
 S4       London                 20   NULL     NULL                 NULL
 S2       Paris                  10   NULL     NULL                 NULL
 S3       Paris                  30   NULL     NULL                 NULL
5 rows selected
```

2.2 Obsolete Features

None for this release.

3 Enhancements And Changes Provided in Oracle Rdb Release 7.4.1.1

3.1 Enhancements

3.1.1 Optional Builtin Function RDB$$IS_ROW_FRAGMENTED

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

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

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

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

Usage Notes

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

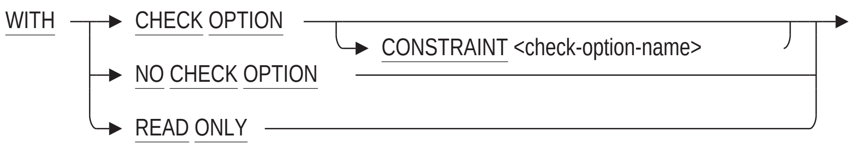
TABLE or ALTER DOMAIN statements that have increased the size of columns.

3.1.2 WITH READ ONLY Clause for CREATE and ALTER VIEW Statements

This release of Oracle Rdb adds support for READ ONLY view definitions, using CREATE VIEW, DECLARE LOCAL TEMPORARY VIEW or ALTER VIEW statements.

Syntax

check-option-clause =



Under normal circumstances, views are considered to be READ ONLY and Rdb will prevent INSERT, UPDATE and DELETE through those views when the select expression uses one of these clauses:

* Includes the DISTINCT operator to eliminate duplicate rows from the result table
* Names more than one table or view in the FROM clause
* Uses a derived table as the row source for a FROM clause
* Includes an aggregate function in the select list
* Contains a UNION, EXCEPT, MINUS, INTERSECT, GROUP BY, or HAVING clause

With this release the database administrator can also force a view to be READ ONLY by applying the WITH READ ONLY clause, even if the factors listed above are not true; i.e. a view that would normally be updatable is considered read-only.

This clause (WITH READ ONLY) and the WITH CHECK OPTION clause are mutually exclusive. Any CHECK OPTION constraint previously defined for the view will be deleted when WITH READ ONLY is used. Conversely if the view is altered to successfully add a CHECK OPTION then the READ ONLY attribute is removed.

The following example shows creating a view on a base table and forcing the view to be read-only.

```
SQL> create view SHOW_CURRENT_SALARY
cont>    as
cont>    select employee_id, salary_amount
cont>    from salary_history
cont>    where salary_end is null
cont>    with read only
cont> ;
```

3.1.3 Enhanced Conversion of Date/time String Literals

In prior releases of Oracle Rdb, string literals assigned to DATE VMS columns, or which were CAST to DATE VMS data types were translated by the SQL interface prior to being passed to the Oracle Rdb Server for execution. This small optimization could reduce or avoid the actual CAST operation.

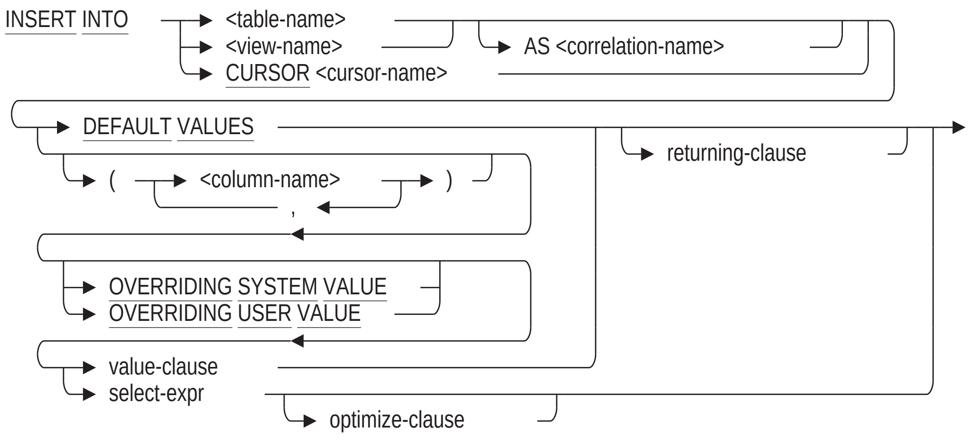
This release of Oracle Rdb expands this support to string literals assigned to DATE ANSI, TIME, TIMESTAMP and INTERVAL data types. If the format of the string is invalid then an error is immediately reported. This is especially a benefit when applications were compiled using the SQL Precompiler or SQL Module Language which now reports the improper format during compile instead of being deferred to runtime.

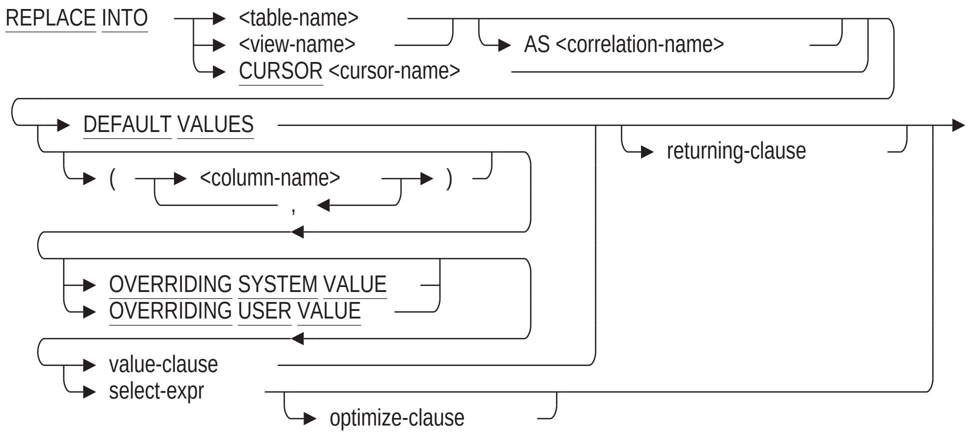
```
SQL> select cast ('1-Jan-2021' as date ansi) from rdb$database;
%SQL-F-DATCONERR, Data conversion error for string '1-Jan-2021'
-COSI-F-IVTIME, invalid date or time
SQL> select cast ('2021-1-1' as date ansi) from rdb$database;

 2021-01-01
1 row selected
SQL>
```

3.1.4 Support for OVERRIDING Clause in INSERT and REPLACE Statements

This release of Oracle Rdb supports the ANSI and ISO SQL Database Language Standard OVERRIDING clause for the INSERT statement. Oracle Rdb also extends this support to the REPLACE statement. The OVERRIDING USER VALUE and OVERRIDING SYSTEM VALUE clauses affect the handling of inserts to any generated columns during INSERT or REPLACE statements. The OVERRIDING clause appears before the VALUES clause or before the SELECT clause as part of the INSERT and REPLACE statements.

Syntax





* The OVERRIDING SYSTEM VALUE clause instructs Rdb that the GENERATED, IDENTITY or AUTOMATIC AS columns will be updated with user supplied values and therefore no generated values will be created. Such a clause would be used if a table was being reloaded after maintenance and the database administrator wanted to retain the saved generated values.

Note

The INSERT or REPLACE statements can use the DEFAULT keyword in place of a column value. When the column being updated is a generated or automatic column then the OVERRIDING clause has no effect on that column as it will be the same in either case.

```
replace into SALES_EMPLOYEES (employee_id, last_name, first_name)
    overriding system value
    values (default, 'Myotte           ', 'Daniel');
```

This clause is similar to the SET FLAGS 'AUTO_OVERRIDE' feature.

The following example shows the use of OVERRIDING SYSTEM VALUE in the case of propagating a daily sales table to the yearly accumulated sales table. In this case we don't want the new generated values for these columns as that was already done by the INSERT into the daily sales table.

```
SQL> --> Now we want to perform end-of-day processing.
SQL> set transaction
cont>     read write
cont>     reserving DAILY_SALES for exclusive write,
cont>           YEARLY_SALES for exclusive write;
SQL>
SQL> --> move daily sales
SQL> insert into YEARLY_SALES
cont>     overriding system value
cont>     select * from DAILY_SALES
cont> ;
7 rows inserted
SQL>
SQL> truncate table DAILY_SALES;
SQL>
SQL> commit;
SQL>
```

* The OVERRIDING USER VALUE clause instructs Rdb that the GENERATED, IDENTITY or AUTOMATIC AS columns will be generated by the database system and that any user supplied values will be ignored. Such a clause would be used when column names were wild carded by a SELECT clause and therefore avoids enumerating all non-generated column names.

  The following example shows copying all daily sales to a log table which will generate new sales_id values.

```
SQL> --> move daily sales and generate new column values for automatic columns
SQL> insert into SALES_LOG
cont>     overriding user value
cont>     select * from DAILY_SALES
cont> ;
```

3.1.5 RMU Recover Progess_report Qualifier and Ctrl-T Display

In this release a Progress_report qualifier has been added to the RMU Recover command. The Progress_report=n qualifier, where n is the time interval in seconds, displays the performance and progress of the database RMU RECOVER operation at timed intervals to SYS$OUTPUT.

The same display can also be output whenever Ctrl-T is typed during the RMU Recover operation. Ctrl-T must have been previously enabled at the DCL level using SET CONTROL=T. SET CONTROL=T requires that SET TERMINAL/BROADCAST is enabled for the display terminal.

The RMU Recover performance and progress display has the following format.

- The first line is the file specification of the after image journal (AIJ) currently being recovered.
- The second line displays the number of megabytes that have been read and processed from the AIJ file during the current interval, the percentage of the AIJ file that has been processed, the number of megabytes that are currently being read per second, and the estimated completion time for the processing of this AIJ file.

The Progress_report interval that has been specified in this case is 1 second.

```
DEVICE:[DIRECTORY]TEST_JOURNAL.AIJ;1
   Read    21 MB (29%) at   21 MB/s, estimated completion time 11:50:29.80
```

The RMU Recover performance and progress display will be different if the Format=NEW_TAPE qualifier has been specified with the RMU Recover command. This is due to limitations caused by the way AIJ data in this format is processed.

- The first line is the file specification of the current temporary AIJ work file being recovered.
- The second line displays the number of megabytes that have been read and processed from the AIJ work file during this interval, and the number of megabytes that are currently being read per second.

The Progress_report interval that has been specified in this case is 1 second.

```
DEVICE:[DIRECTORY]AIJ_WORKG8V0RS6M99D1GJKG4I80.AIJ;
   Read    36 MB at   36 MB/s
```

Syntax

The command line syntax for the RMU Recover command Progress_report qualifier is

```
/PROGRESS_REPORT[=seconds]
```

This qualifier cannot be negated and is not the default. The default value for the Progress_report display interval is 60 seconds. The minimal value for the Progress_report display interval is 1 second.

Examples

The following example shows the recovery of one backed up AIJ file with a progress report interval of 1 second. A zero value indicates no data was read from the AIJ file during that interval. The completion time estimate is an approximation not guaranteed to be exact.

```
$ SHOW TIME
  11-AUG-2020 16:04:39
$ rmu/recover/LOG/root=DEVICE:[DIRECTORY]glory.rdb/PROGRESS_REPORT=1 -
  DEVICE:[DIRECTORY]backup_after.baij
%RMU-I-LOGRECDB, recovering database file DEVICE:[DIRECTORY]GLORY.RDB;1
%RMU-I-LOGOPNAIJ, opened journal file DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
 at 11-AUG-2020 16:04:39.19
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read    10 MB ( 7%) at   10 MB/s, estimated completion time 16:04:52.75
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read    19 MB (13%) at    9 MB/s, estimated completion time 16:04:55.39
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read    52 MB (35%) at   32 MB/s, estimated completion time 16:04:45.16
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read    74 MB (49%) at   21 MB/s, estimated completion time 16:04:46.57
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read    74 MB (49%) at    0 KB/s, estimated completion time 16:04:44.19
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read    74 MB (49%) at    0 KB/s, estimated completion time 16:04:45.19
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read    74 MB (49%) at    0 KB/s, estimated completion time 16:04:46.19
%RMU-I-LOGRECSTAT, transaction with TSN 225 committed
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read    78 MB (52%) at    3 MB/s, estimated completion time 16:05:04.80
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read   100 MB (68%) at   22 MB/s, estimated completion time 16:04:50.26
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read   118 MB (79%) at   17 MB/s, estimated completion time 16:04:50.88
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read   141 MB (95%) at   23 MB/s, estimated completion time 16:04:50.45
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read   148 MB (99%) at    6 MB/s, estimated completion time 16:04:51.20
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read   148 MB (99%) at    0 KB/s, estimated completion time 16:04:52.19
DEVICE:[DIRECTORY]BACKUP_AFTER.BAIJ;1
   Read   148 MB (99%) at    0 KB/s, estimated completion time 16:04:53.19
%RMU-I-LOGRECSTAT, transaction with TSN 226 committed
%RMU-I-AIJONEDONE, AIJ file sequence 0 roll-forward operations completed
%RMU-I-LOGRECOVR, 2 transactions committed
%RMU-I-LOGRECOVR, 0 transactions rolled back
%RMU-I-LOGRECOVR, 0 transactions ignored
%RMU-I-AIJNOACTIVE, there are no active transactions
%RMU-I-AIJSUCCES, database recovery completed successfully
%RMU-I-AIJNXTSEQ, to continue this AIJ file recovery, the sequence number
 needed will be 1
%RMU-I-AIJALLDONE, after-image journal roll-forward operations completed
```

```
%RMU-I-LOGSUMMARY, total 2 transactions committed
%RMU-I-LOGSUMMARY, total 0 transactions rolled back
%RMU-I-LOGSUMMARY, total 0 transactions ignored
%RMU-I-AIJSUCCES, database recovery completed successfully
%RMU-I-AIJFNLSEQ, to start another AIJ file recovery, the sequence number
 needed will be 1
%RMU-I-AIJNOENABLED, after-image journaling has not yet been enabled
$ SHOW TIME
  11-AUG-2020 16:04:53
```

The following example shows the recovery of two AIJ files with a progress report interval of 1 second. A zero value indicates no data was read from the AIJ file during that interval. The completion time estimate is an approximation not guaranteed to be exact.

```
$ SHOW TIME
  11-AUG-2020 16:08:20
$ rmu/recover/root=DEVICE:[DIRECTORY]glory.rdb/PROGRESS_REPORT=1 -
DEVICE:[DIRECTORY]backup_after1.aij, -
DEVICE:[DIRECTORY]backup_after2.aij
%RMU-I-LOGRECDB, recovering database file DEVICE:[DIRECTORY]GLORY.RDB;1
%RMU-I-LOGOPNAIJ, opened journal file DEVICE:[DIRECTORY]BACKUP_AFTER1.AIJ;1
 at 11-AUG-2020 16:08:20.90
DEVICE:[DIRECTORY]BACKUP_AFTER1.AIJ;1
   Read    33 MB (44%) at   33 MB/s, estimated completion time 16:08:23.12
DEVICE:[DIRECTORY]BACKUP_AFTER1.AIJ;1
   Read    64 MB (86%) at   30 MB/s, estimated completion time 16:08:23.23
DEVICE:[DIRECTORY]BACKUP_AFTER1.AIJ;1
   Read    74 MB (99%) at    9 MB/s, estimated completion time 16:08:23.92
DEVICE:[DIRECTORY]BACKUP_AFTER1.AIJ;1
   Read    74 MB (99%) at    0 KB/s, estimated completion time 16:08:24.90
DEVICE:[DIRECTORY]BACKUP_AFTER1.AIJ;1
   Read    74 MB (99%) at    0 KB/s, estimated completion time 16:08:25.90
%RMU-I-LOGRECSTAT, transaction with TSN 225 committed
%RMU-I-LOGRECSTAT, transaction with TSN 227 committed
%RMU-I-AIJONEDONE, AIJ file sequence 0 roll-forward operations completed
%RMU-I-LOGRECOVR, 2 transactions committed
%RMU-I-LOGRECOVR, 0 transactions rolled back
%RMU-I-LOGRECOVR, 0 transactions ignored
%RMU-I-AIJACTIVE, 1 active transaction not yet committed or aborted
%RMU-I-LOGRECSTAT, transaction with TSN 226 is active
%RMU-I-AIJSUCCES, database recovery completed successfully
%RMU-I-AIJNXTSEQ, to continue this AIJ file recovery, the sequence number
 needed will be 1
%RMU-I-LOGOPNAIJ, opened journal file DEVICE:[DIRECTORY]BACKUP_AFTER2.AIJ;1
 at 11-AUG-2020 16:08:26.63
DEVICE:[DIRECTORY]BACKUP_AFTER2.AIJ;1
   Read    32 MB (44%) at   32 MB/s, estimated completion time 16:08:28.90
DEVICE:[DIRECTORY]BACKUP_AFTER2.AIJ;1
   Read    67 MB (90%) at   34 MB/s, estimated completion time 16:08:28.82
DEVICE:[DIRECTORY]BACKUP_AFTER2.AIJ;1
   Read    73 MB (99%) at    6 MB/s, estimated completion time 16:08:29.68
DEVICE:[DIRECTORY]BACKUP_AFTER2.AIJ;1
   Read    73 MB (99%) at    0 KB/s, estimated completion time 16:08:30.63
DEVICE:[DIRECTORY]BACKUP_AFTER2.AIJ;1
   Read    73 MB (99%) at    0 KB/s, estimated completion time 16:08:31.63
%RMU-I-LOGRECSTAT, transaction with TSN 226 committed
%RMU-I-AIJONEDONE, AIJ file sequence 1 roll-forward operations completed
%RMU-I-LOGRECOVR, 1 transaction committed
%RMU-I-LOGRECOVR, 0 transactions rolled back
%RMU-I-LOGRECOVR, 0 transactions ignored
%RMU-I-AIJNOACTIVE, there are no active transactions
%RMU-I-AIJSUCCES, database recovery completed successfully
%RMU-I-AIJNXTSEQ, to continue this AIJ file recovery, the sequence number
 needed will be 2
```

```
%RMU-I-AIJALLDONE, after-image journal roll-forward operations completed
%RMU-I-LOGSUMMARY, total 3 transactions committed
%RMU-I-LOGSUMMARY, total 0 transactions rolled back
%RMU-I-LOGSUMMARY, total 0 transactions ignored
%RMU-I-AIJSUCCES, database recovery completed successfully
%RMU-I-AIJFNLSEQ, to start another AIJ file recovery, the sequence number
 needed will be 2
%RMU-I-AIJNOENABLED, after-image journaling has not yet been enabled
$ SHOW TIME
  11-AUG-2020 16:08:31
```

3.1.6 Additional Options for the RMU Set Database Command

This release of Oracle Rdb adds the following qualifiers to the RMU Set Database statement. This command is an OFFLINE command and requires exclusive access to the target database.

- /NODES_MAX

  Sets the number of nodes that are permitted to attach to the database. This command is equivalent to the SQL ALTER DATABASE ... NUMBER OF CLUSTER NODES statement.

  This qualifier can only be applied to multi-file databases. RMU will report an error for single file databases. Use SQL EXPORT DATABASE and IMPORT DATABASE to change this value for a single file database.

  ```
  $ rmu/set data/node=1 personnel
  %RMU-F-MFDBONLY, operation is not allowed on single-file databases
  %RMU-F-FTL_RMU, Fatal error for RMU operation at 29-OCT-2020 13:55:55.33
  ```

- /RESERVE

  This clause alters the reserve limit of the database. One or more of the keywords AREAS, CACHES, JOURNALS, or SEQUENCES with new values can be specified.

  This qualifier can only be applied to multi-file databases. RMU will report an error for single file databases. Use SQL EXPORT DATABASE and IMPORT DATABASE to change this value for a single file database.

  ```
  $ rmu/set data/res=area=10 personnel
  %RMU-F-MFDBONLY, operation is not allowed on single-file databases
  %RMU-F-FTL_RMU, Fatal error for RMU operation at 29-OCT-2020 13:59:19.94
  ```

  - AREAS

    Reserves extra storage area entries to allow subsequent ALTER DATABASE ... ADD STORAGE AREA statements.

  - CACHES

    Reserves extra row cache entries to allow subsequent ALTER DATABASE ... ADD CACHE statements.

  - JOURNALS

    Reserves extra after image journal entries to allow subsequent ALTER DATABASE ... ADD JOURNAL statements, or RMU Set After_Journal command.

  - SEQUENCES

Reserves sequence entries. This action should be taken when a CREATE TABLE with IDENTITY or a CREATE SEQUENCE statement fails due to insufficient sequence table entries; RDMS-F-SEQTBLFUL, sequence table is full.

```
SQL>  create sequence NEW_PRODUCT_CODES;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-SEQTBLFUL, sequence table is full
SQL>
```

- /USERS_MAX

  Sets the number of users that are permitted to attach to the database. This command is equivalent to the SQL ALTER DATABASE ... NUMBER OF USERS statement.

  This qualifier can only be applied to multi-file databases. RMU will report an error for single file databases. Use SQL EXPORT DATABASE and IMPORT DATABASE to change this value for a single file database.

  ```
  $ rmu/set data/users=11 personnel
  %RMU-F-MFDBONLY, operation is not allowed on single-file databases
  %RMU-F-FTL_RMU, Fatal error for RMU operation at 29-OCT-2020 13:58:29.03
  ```

3.1.7 New Summary_only Qualifier to RMU Dump Audit Command

This release of Oracle Rdb adds the SUMMARY_ONLY qualifier to RMU Dump Audit. This allows the database administrator to see a list of databases that have entries recorded in the named AUDIT$JOURNAL.

Neither the /FORMAT nor the /TYPE qualifiers are permitted with /SUMMARY_ONLY. The database parameter is ignored.

The following example generates a file containing the database names used by that version of the SECURITY.AUDIT$JOURNAL.

```
$ define/nolog RMU_AJ SYS$COMMON:[SYSMGR]SECURITY.AUDIT$JOURNAL;8398
$ rmu/dump/audit -
    "" -
    RMU_AJ -
    /since=TODAY -
    /log -
    /summary_only -
    /output=audit_dump.txt
$
```

3.1.8 New RMU Verify Minimize_conflicts Qualifier

In this release of Oracle Rdb, a /MINIMIZE_CONFLICTS qualifier has been added to the RMU Verify command. The /MINIMIZE_CONFLICTS qualifier attempts to reduce database page lock conflicts at timed intervals during the verify operation.

Syntax

- /MINIMIZE_CONFLICTS[=n]

  This qualifier requests that RMU Verify periodically attempt to reduce locks on buffers during operation. The value of (n) is the time interval in seconds. The minimal value that can be specified is 1 second. The default time interval is 30 seconds.
- /NOMINIMIZE_CONFLICTS

  This qualifier requests that RMU Verify request that RMU not release locking periodically. This returns RMU to prior default behavior.
- If this qualifier is omitted then the default is assumed to be MINIMIZE_ CONFLICTS=30

Examples

The following example shows examples for using this new MINIMIZE_ CONFLICTS feature.

```
$!
$! Default - minimize database page lock conflicts at 30 second
$! intervals
$!
$ RMU/VERIFY/ALL/NOLOG TEST.RDB
$!
$! Default time interval - minimize database page lock conflicts
$! at 30 second intervals
$!
$ RMU/VERIFY/ALL/NOLOG/MINIMIZE_CONFLICTS TEST.RDB
$!
$! Minimize database page lock conflicts at 1 second intervals
$!
$ RMU/VERIFY/ALL/NOLOG/MINIMIZE_CONFLICTS=1 TEST.RDB
$!
$! Do not minimize page lock conflicts
$!
$ RMU/VERIFY/ALL/NOLOG/NOMINIMIZE_CONFLICTS TEST.RDB
$
```

3.1.9 New OPTION=GENERATED Added to RMU Extract Command

This release of Oracle Rdb includes a new GENERATED option for RMU Extract. In prior releases RMU Extract ITEM=UNLOAD and ITEM=LOAD would generate load commands that assumed all the columns were updatable.

The option FULL can be used to generate syntax that loads every field by name and includes virtual columns (AUTOMATIC AS, GENERATED, IDENTITY and COMPUTED BY) as commented out names. Therefore, editing was required to uncomment GENERATED column names so they could be reloaded. In addition the /VIRTUAL=AUTOMATIC qualifier needed to be added to the RMU Load and RMU Unload commands.

Now using OPTION=(GENERATED) will instruct RMU Extract to generate more appropriate DCL commands for unloading and re-loading data in tables that contain GENERATED columns.

The following example shows a portion of a generated DCL procedure when only OPTION=FULL is used.

```
$ RMU/EXTRACT/ITEM=UNLOAD/OPTION=FULL SAMPLE_DB
   .
   .
   .
$ CREATE SAMPLE0.COLUMNS
! Columns list for table SAMPLE0
! in ...
! Created by RMU Extract for Oracle Rdb ... on 29-JAN-2021 13:20:28.40
! Virtual: IDENT_COL
DETAILS
! Virtual: LAST_UPDATE
$ RMU/UNLOAD -
    USER1:[TESTING.DATABASES]MF_PERSONNEL_SQL.RDB -
    /FIELDS="@SAMPLE0.COLUMNS" -
    SAMPLE0 -
    SAMPLE0.UNL
$
```

The following example shows a portion of a generated DCL procedure when OPTION=(GENERATED) is used.

```
$ RMU/EXTRACT/ITEM=UNLOAD/OPTION=GENERATED SAMPLE_DB
   .
   .
   .
$ CREATE SAMPLE0.COLUMNS
! Columns list for table SAMPLE0
! in ...
! Created by RMU Extract for Oracle Rdb ... on 29-JAN-2021 13:23:27.76
IDENT_COL
DETAILS
LAST_UPDATE
$ RMU/UNLOAD -
    USER1:[TESTING.DATABASES]MF_PERSONNEL_SQL.RDB -
    /FIELDS="@SAMPLE0.COLUMNS" -
    /VIRTUAL=AUTOMATIC -
    SAMPLE0 -
    SAMPLE0.UNL
$
```

3.1.10 Changed Behavior for the Noedit_filename Qualifier in RMU Backup After_Journal Command

In prior releases of Oracle Rdb the /NOEDIT_FILENAME qualifier on the RMU Backup After_Journal was ignored. With this release it takes on a new meaning as described below:

– /EDIT_FILENAME

  As with previous versions, this qualifier defines the editing to be performed for the output backup file name. This editing is performed on the provided backup filename, or if "" is specified the default backup filename defined in the database.

  This qualifier replaces any EDIT_FILENAME defined for the database.

– /NOEDIT_FILENAME

This qualifier negates any prior usage on the command of the /EDIT_FILENAME qualifier and also instructs RMU to ignore the EDIT_FILENAME defined by the SQL ALTER DATABASE statement, or RMU Set After_Journal command. This is a change of behavior from prior versions and supports the enhancements made to the RMU Set After_Journal command which allows the defaults to be defined for the MANUAL backup processing.

No editing is performed on the provided backup filename, or if "" is specified the default backup filename defined in the database is used without changes.

– Neither /EDIT_FILENAME nor /NOEDIT_FILENAME was used.

  In this case RMU Backup After_Journal will use the default if defined in the database by SQL ALTER DATABASE statement, or RMU Set After_Journal command.

3.2 Obsolete Features

3.2.1 Comma Statement Separator in Trigger Body No Longer Supported

The syntax for trigger actions in the CREATE TRIGGER statement has, in the past, supported the comma (,) as well as the semicolon (;) as statement separators. The use of the comma separator has been problematic in Oracle Rdb SQL because it conflicts in various places with the comma used as an element separator within some statements. For example, the TRACE statement allows a comma separated list of values and the INSERT INTO ... SELECT ... FROM statement allows a comma separated list of table names in the FROM clause. In these cases, a comma cannot be used as a statement separator because the current statement appears to be continued.

Future versions of Oracle Rdb are expected to include enhancements to the TRIGGER action syntax which will allow other statements to include comma as an element separator. Therefore, the comma statement separator is now no longer supported.

Any scripts or applications that include the CREATE TRIGGER statement must now be modified to use only the semicolon (;) as a separator.

This change does not affect existing database triggers, only new triggers defined using the CREATE TRIGGER statement. The RMU Extract Item=TRIGGER command already generates semicolon separators in extracted CREATE TRIGGER statements.

4 Enhancements And Changes Provided in Oracle Rdb Release 7.4.1.0

4.1 Enhancements

4.1.1 PCSI Support for Rdb Kit Installation and Deinstallation

Whenever Oracle Rdb is installed or deinstalled, Oracle Rdb will be registered in the PCSI software product database. This will allow users to use the PCSI PRODUCT SHOW HISTORY and PRODUCT SHOW PRODUCT commands to display information about releases of Oracle Rdb that have been installed or deinstalled. This information will also be helpful as input whenever a Service Request (SR) is submitted to Oracle Support.

The following lines will now be displayed during the installation of Oracle Rdb, showing that the installation has been registered in the PCSI database.

```
The following product has been selected:
    ORCL I64VMS RDB74 V7.4-100              Transition (registration)

The following product will be registered:
    ORCL I64VMS RDB74 V7.4-100              DISK$NODE84_2:[VMS$COMMON.]

File lookup pass starting ...

Portion done: 0%
...100%

File lookup pass completed search for all files listed in the product's PDF
Total files searched: 0   Files present: 0   Files absent: 0

The following product has been registered:
    ORCL I64VMS RDB74 V7.4-100             Transition (registration)
%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...
```

Registration in the PCSI software product database allows a user to use commands such as the following to track what Oracle Rdb releases are currently installed and the history of any past product installations and deinstallations.

```
$ PRODUCT SHOW HISTORY/SINCE
------------------------------------ ----------- ----------- --- -----------
PRODUCT                              KIT TYPE    OPERATION   VAL DATE
------------------------------------ ----------- ----------- --- -----------
ORCL I64VMS RDB74 V7.4-100           Transition  Reg Product (U) 10-JUN-2020
------------------------------------ ----------- ----------- --- -----------

1 item found

$ PRODUCT SHOW HISTORY RDB7*
------------------------------------ ----------- ----------- --- -----------
PRODUCT                              KIT TYPE    OPERATION   VAL DATE
------------------------------------ ----------- ----------- --- -----------
ORCL I64VMS RDB74 V7.4-100           Transition  Reg Product (U) 10-JUN-2020
------------------------------------ ----------- ----------- --- -----------

1 item found
```

```
$ PRODUCT SHOW PRODUCT RDB7*
----------------------------------- ----------- ---------
PRODUCT                             KIT TYPE    STATE
----------------------------------- ----------- ---------
ORCL I64VMS RDB74 V7.4-100          Transition  Installed
----------------------------------- ----------- ---------

1 item found
```

The following lines will now be displayed during the deinstallation of Oracle Rdb, showing that the removal of the release has been registered in the PCSI database. Deinstallation is performed by executing the DCL procedure SYS$MANAGER:RDB$DEINSTALL_DELETE.COM. Please refer to section "Deleting Versions of Oracle Rdb" in the Oracle Rdb Installation Guide for further details.

```
The following product has been selected:
    ORCL I64VMS RDB74 V7.4-100             Transition (registration)

The following product will be removed from destination:
    ORCL I64VMS RDB74 V7.4-100             DISK$CLYPPR84_2:[VMS$COMMON.]

Portion done: 0%...100%

The following product has been removed:
    ORCL I64VMS RDB74 V7.4-100             Transition (registration)
```

The example below shows the additional information that will be displayed by the PCSI PRODUCT commands as a result of the deinstallation of a release of Oracle Rdb.

```
$ PRODUCT SHOW HISTORY/SINCE
----------------------------------- ----------- ----------- --- -----------
PRODUCT                             KIT TYPE    OPERATION   VAL DATE
----------------------------------- ----------- ----------- --- -----------
ORCL I64VMS RDB74 V7.4-100          Transition  Remove       -  10-JUN-2020
ORCL I64VMS RDB74 V7.4-100          Transition  Reg Product (U) 10-JUN-2020
----------------------------------- ----------- ----------- --- -----------
2 items found

$ PRODUCT SHOW HISTORY RDB7*
----------------------------------- ----------- ----------- --- -----------
PRODUCT                             KIT TYPE    OPERATION   VAL DATE
----------------------------------- ----------- ----------- --- -----------
ORCL I64VMS RDB74 V7.4-100          Transition  Remove       -  10-JUN-2020
ORCL I64VMS RDB74 V7.4-100          Transition  Reg Product (U) 10-JUN-2020
----------------------------------- ----------- ----------- --- -----------
2 items found

$ PRODUCT SHOW PRODUCT RDB7*
----------------------------------- ----------- ---------
PRODUCT                             KIT TYPE    STATE
----------------------------------- ----------- ---------
0 items found
```

4.1.2 Some Aggregate Functions Inherit Source Column EDIT STRING

Oracle Rdb supports EDIT STRING inheritance for these functions when using Interactive SQL.

- MAX, MEDIAN, MIN, FIRST_VALUE, LAST_VALUE

  When the input type matches the output type, then the EDIT STRING from the source column is inherited to improve the readability of the aggregate.

- CAST

  When the datatype of the CAST includes a domain with the EDIT STRING.

The following example shows the EDIT STRING being used.

```
SQL> create domain DOM_TST integer(2) edit string '$(9)9.99';
SQL>
SQL> create table TST
cont>     (a integer(2) edit string '$(9)9.99'
cont>     ,c char(10)
cont>     );
SQL>
SQL> insert into TST
cont>     values (100, 100, 'A');
1 row inserted
SQL> insert into TST
cont>     values (233, 233, 'B');
1 row inserted
SQL>
SQL> --> column with explicit edit string
SQL> select min (a), max (a), cast (a as DOM_TST)
cont>  from TST
cont>  group by a
cont> ;
cont> ;

       $100.00         $100.00         $100.00
       $233.00         $233.00         $233.00
2 rows selected
SQL>
SQL> select first_value (a) within group (order by b desc),
cont>        last_value (a) within group (order by b desc),
cont>        median (a)
cont>  from TST
cont> ;

       $233.00         $100.00         $166.50
1 row selected
SQL>
```

Use the SET DISPLAY NO EDIT STRING statement to disable this behavior.

4.1.3 Enhanced LIKE Table Support in CREATE TABLE Statement

This release of Oracle Rdb introduces support for the ANSI and ISO SQL Language Standard syntax for the LIKE table clause. It also adds new EXCLUDING and INCLUDING clauses to the LIKE clause within the CREATE TABLE statement.

In prior releases of Oracle Rdb, a table can be created using syntax similar to the following:

```
SQL> create table RETIRED_EMPLOYEES
cont>     like EMPLOYEES
cont>     ;
SQL>
```

This statement copies the definitions of each column as well as DEFAULT values defined for those source columns. SQL also allows additional columns and constraints to be defined for the new table.

```
SQL> create table RETIRED_EMPLOYEES
cont>     like EMPLOYEES
cont>     (retirement_date  DATE
cont>     ,check (retirement_date > birthday) not deferrable
cont>     );
SQL>
```

This syntax is retained for backward compatibility with prior releases of Oracle Rdb.

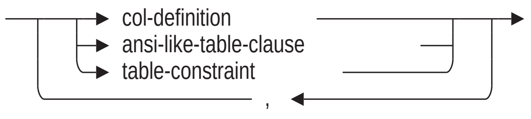
The syntax for a similar feature in the ANSI/ISO SQL Database Language moves the LIKE clause into the section that defines the columns and constraint. This adds the ability to copy column definitions from more than one table, control how GENERATED, AUTOMATIC, IDENTITY and COMPUTED columns are inherited, as well as define the column ordering; this is determined by the order of the listed columns and tables.

```
SQL> create table RETIRED_EMPLOYEES
cont>     (retirement_date  DATE
cont>     ,like EMPLOYEES
cont>         including COMPUTED
cont>         excluding DEFAULTS
cont>     ,check (retirement_date > birthday) not deferrable
cont>     ,unique (employee_id)
cont>     ,hr_authorizations LIST OF BYTE VARYING
cont>     );
SQL>
```

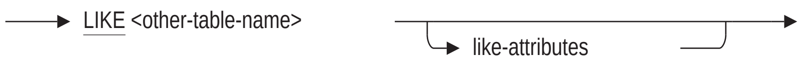
By default, GENERATED, AUTOMATIC, IDENTITY and COMPUTED columns are not copied but columns representing the same data types are created instead.

Syntax

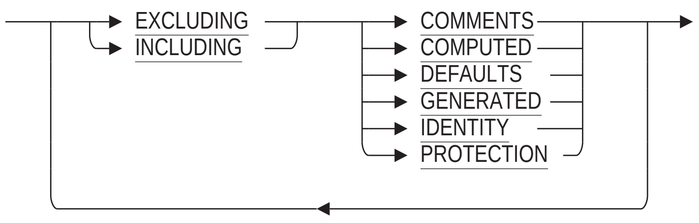
column-constraint-list =



ansi-like-table-clause =



like-attributes =



Usage Notes

- When using the LIKE clause to copy a table definition, the creator of the new table must have REFERENCES or SELECT privilege granted for the referenced table.
- By default, Rdb includes the column protections (access control lists) and comments for any copied column. These new clauses allow the database administrator to suppress the copying of that metadata.

- The LIKE clause can be used multiple times within a CREATE TABLE statement. However, if the copied tables include any duplicate column names, then an error will be reported. Only one IDENTITY column can be defined or inherited. Use the INCLUDING IDENTITY clause, if necessary, to inherit the attributes from the referenced table.

  The default behavior is EXCLUDING COMPUTED, GENERATED, IDENTITY column details. In this case, non-generated columns will be created which contain the same data type attributes. Default values defined for the source tables are not automatically inherited. Use the INCLUDING DEFAULTS clause to control this behavior.

  Note: For backward compatibility with previous versions of Oracle Rdb, the LIKE clause used outside the column-constraint-list defaults to INCLUDING GENERATED, INCLUDING IDENTITY, INCLUDING COMPUTED and INCLUDING DEFAULTS. The like-attributes may not be specified in this location and therefore these defaults may not be changed.
- The clauses EXCLUDING GENERATED or INCLUDING GENERATED apply to columns defined using the GENERATED ... AS (expr) and AUTOMATIC ... AS (expr) syntax. When EXCLUDING is used or implied, the generated (or automatic) column is converted to a simple base column with the same data types.
- The clauses EXCLUDING IDENTITY or INCLUDING IDENTITY apply to columns defined using the GENERATED ... AS IDENTITY and IDENTITY (...) syntax. When EXCLUDING is used or implied, the identity column is converted to a simple base column with the same data types.
- The clauses EXCLUDING COMPUTED or INCLUDING COMPUTED apply to columns defined using the COMPUTED BY expr syntax. When EXCLUDING is used or implied, the computed by column is converted to a simple base column with the same data types. Note that the column will require space in the defined table, which isn't true for COMPUTED BY columns.
- When the LIKE clause is used within the column-constraint-list, then EXCLUDING DEFAULTS is assumed. Use the INCLUDING DEFAULTS if you wish the inherited columns to have DEFAULTS inherited from the source table.
- The LIKE clause is only used to inherit the column definitions from the referenced table. Once the table is created with LIKE clauses, subsequent changes to the source table are not propagated to the created tables.

Examples

The following example shows the use of the LIKE clause to inherit columns from various template tables.

```
SQL> create table NAMES_REC
cont>     (LAST_NAME          LAST_NAME_DOM
cont>     ,FIRST_NAME         FIRST_NAME_DOM
cont>     ,MIDDLE_INITIAL     MIDDLE_INITIAL_DOM
cont>     );
SQL>
SQL> create table ADDRESS_REC
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>     );
SQL>
SQL> create table employees
cont>     (EMPLOYEE_ID        ID_DOM not null
cont>     ,like NAMES_REC     including DEFAULTS
cont>     ,like ADDRESS_REC   including DEFAULTS
cont>     ,SEX                SEX_DOM
cont>     ,BIRTHDAY           DATE_DOM
cont>     ,STATUS_CODE        STATUS_CODE_DOM
cont>     );
SQL>
```

The resulting CREATE TABLE for the EMPLOYEES table is easier to read and allows for consistency among similar definitions.

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

Columns for table EMPLOYEES:
Column Name                     Data Type        Domain
-----------                     ---------        ------
EMPLOYEE_ID                     CHAR(5)          ID_DOM
 Not Null constraint EMPLOYEES_EMPLOYEE_ID_NOT_NULL
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

SQL>
```

4.1.4 RMU Reclaim Free_pages Qualifier Frees Unused Data Page Clumps

There is an additional Free_pages qualifier for the RMU Reclaim command. This qualifier is used to free unused data page clumps that are allocated in uniform storage areas. It will free all unused page clumps in an entire uniform storage area or all unused page clumps in one or more specified table or index logical areas in uniform storage areas. Any deleted dbkeys and locked space on pages will also be freed.

Command Qualifiers

/[NO]FREE_PAGES

Nofree_pages is the default.

Other qualifiers may be used in conjunction with the Free_pages qualifier.

/AREA[=storage-area-name-list]

Area is used to specify a list of uniform storage area names to process. The wild card syntax AREA=* can be specified for processing all uniform storage areas in the database.

The default for the Area qualifier is all uniform storage areas in the database.

/LAREA=logical-area-name-list

Larea is used to specify a list of individual table or index logical area names to process.

There is no default for the Larea qualifier. A list of logical area names must be specified. The logical area name will be used to determine the storage area where the logical area is located. If the logical area is partitioned among multiple storage areas, each logical area partition will be processed.

This qualifier can only be specified if the Free_pages qualifier is specified.

/LOCK_TIMEOUT=seconds

Lock_timeout is used to specify a lock timeout value that will be in effect during the execution of the RMU Reclaim Free_pages command.

Lock_timeout can only be specified if Free_pages is also specified. The value specified with this qualifier is the maximum time in seconds during which the current RMU Reclaim Free_pages command will wait to acquire an exclusive update lock on the current storage area or logical area to be processed when accessing an on-line database with other users.

If Lock_timeout is not specified, one of the following values will be used, in the specified order of precedence.

1. The value of the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL, if it has been specified.
2. The "LOCK TIMEOUT INTERVAL" specified by the SQL CREATE or ALTER DATABASE command is used.
3. The RMU Reclaim Free_pages command will wait indefinitely to acquire an exclusive update lock on the current storage area or logical area to be processed.

If /LOCK_TIMEOUT=0 is specified, the RMU Reclaim Free_pages command will ignore any lock timeout defaults that may be in effect and wait indefinitely to acquire an exclusive update lock on the current storage area or logical area to be processed.

Usage Notes

- The Free_pages command can be used when the database is active. Please note that RMU will lock affected areas during processing, which may reduce concurrency.
- Free_pages is not a default qualifier for the RMU Reclaim command. If the Free_pages qualifier is not specified, the RMU Reclaim command will implement the default functionality of freeing deleted dbkeys and locked space in mixed and uniform database storage areas.

- RMU Reclaim Free_Pages can be interrupted at any time; any work in progress will be rolled back. Actions that are completed will have each been committed: if processing a list of logical area names (/LAREA), a commit is performed after each logical area and if processing a list of storage areas (/AREA), a commit is performed after each storage area. Note that tables and indices which are partitioned have multiple logical areas that share the same name as the table or index.
- If Free_pages is specified without either the Larea or Area qualifier, all the uniform storage areas in the database will be processed.
- The Area and Larea qualifiers cannot both be specified in the same RMU Reclaim command.
- If a mixed storage area name is specified with the Area qualifier or the name of a logical area in a mixed storage area is specified with the Larea qualifier, a warning message will be output and a warning status will be returned by the Reclaim command. That storage or logical area will not be processed but the Reclaim command will continue processing the next storage or logical area in the specified list of storage areas or logical areas.
- If a lock wait timeout occurs, a warning message will be output and a warning status will be returned by the Reclaim command. That storage or logical area will not be processed but the Reclaim command will continue processing the next storage area or logical area in the specified list of storage or logical areas.
- The RMU Reclaim Free_pages functionality replaces that provided by RMU REPAIR /INITIALIZE=FREE_PAGES. The main advantage of Reclaim is that it can be run on an active database.

Examples

Examples using /AREA

The following examples show the Free_pages qualifier with the Area qualifier to free unused page clumps for one or more named storage areas.

```
$ RMU/RECLAIM/LOG/AREA=ABM_AREA1/FREE_PAGES ABM_SAMPLE.RDB
%RMU-I-RCLMAREA, Reclaiming area ABM_AREA1
%RMU-I-RCLMPAGPRC, 2138 pages processed for area ABM_AREA1
%RMU-I-RCLMPAGFREED, 1992 clump pages freed for area ABM_AREA1
$
$ RMU/RECLAIM/FREE_PAGES/AREA=(MFDBA2,MFDBA1)/LOG MFDB
%RMU-I-RCLMAREA, Reclaiming area MFDBA2
%RMU-I-RCLMPAGPRC, 13 pages processed for area MFDBA2
%RMU-I-RCLMPAGFREED, 4 clump pages freed for area MFDBA2
%RMU-I-RCLMAREA, Reclaiming area MFDBA1
%RMU-I-RCLMPAGPRC, 13 pages processed for area MFDBA1
%RMU-I-RCLMPAGFREED, 4 clump pages freed for area MFDBA1
$
$ RMU/RECLAIM/FREE_PAGES/AREA=*/LOG MFDB
%RMU-I-RCLMAREA, Reclaiming area DISK:[DIRECTORY]MFDB.RDA;1
%RMU-I-RCLMPAGPRC, 701 pages processed for area
DISK:[DIRECTORY]MFDB.RDA;1
%RMU-I-RCLMPAGFREED, 220 clump pages freed for area
DISK:[DIRECTORY]MFDB.RDA;1
%RMU-I-RCLMAREA, Reclaiming area DISK:[DIRECTORY]MFDBA1.RDA;1
%RMU-I-RCLMPAGPRC, 13 pages processed for area
DISK:[DIRECTORY]MFDBA1.RDA;1
%RMU-I-RCLMPAGFREED, 0 clump pages freed for area
DISK:[DIRECTORY]MFDBA1.RDA;1
%RMU-I-RCLMAREA, Reclaiming area DISK:[DIRECTORY]MFDBA2.RDA;1
```

```
%RMU-I-RCLMPAGPRC, 13 pages processed for area
DISK:[DIRECTORY]MFDBA2.RDA;1
%RMU-I-RCLMPAGFREED, 0 clump pages freed for area
DISK:[DIRECTORY]MFDBA2.RDA;1
%RMU-I-RCLMAREA, Reclaiming area DISK:[DIRECTORY]MFDBA3.RDA;1
%RMU-I-RCLMPAGPRC, 13 pages processed for area
DISK:[DIRECTORY]MFDBA3.RDA;1
%RMU-I-RCLMPAGFREED, 0 clump pages freed for area
DISK:[DIRECTORY]MFDBA3.RDA;1
$
$ RMU/RECLAIM/FREE_PAGES/AREA/LOG MFDB
%RMU-I-RCLMAREA, Reclaiming area DISK:[DIRECTORY]MFDB.RDA;1
%RMU-I-RCLMPAGPRC, 701 pages processed for area
DISK:[DIRECTORY]MFDB.RDA;1
%RMU-I-RCLMPAGFREED, 220 clump pages freed for area
DISK:[DIRECTORY]MFDB.RDA;1
%RMU-I-RCLMAREA, Reclaiming area DISK:[DIRECTORY]MFDBA1.RDA;1
%RMU-I-RCLMPAGPRC, 13 pages processed for area
DISK:[DIRECTORY]MFDBA1.RDA;1
%RMU-I-RCLMPAGFREED, 0 clump pages freed for area
DISK:[DIRECTORY]MFDBA1.RDA;1
%RMU-I-RCLMAREA, Reclaiming area DISK:[DIRECTORY]MFDBA2.RDA;1
%RMU-I-RCLMPAGPRC, 13 pages processed for area
DISK:[DIRECTORY]MFDBA2.RDA;1
%RMU-I-RCLMPAGFREED, 0 clump pages freed for area
DISK:[DIRECTORY]MFDBA2.RDA;1
%RMU-I-RCLMAREA, Reclaiming area DISK:[DIRECTORY]MFDBA3.RDA;1
%RMU-I-RCLMPAGPRC, 13 pages processed for area
DISK:[DIRECTORY]MFDBA3.RDA;1
%RMU-I-RCLMPAGFREED, 0 clump pages freed for area
DISK:[DIRECTORY]MFDBA3.RDA;1
$
```

Examples using /LAREA

The following examples show the Free_pages qualifier with the Larea qualifier to free unused page clumps for one or more named table and index logical areas.

```
$ RMU/RECLAIM/LOG/LAREA=SAMPLE_TABLE/FREE_PAGES ABM_SAMPLE.RDB
%RMU-I-RCLMLAREA, Reclaiming logical area SAMPLE_TABLE in physical
area DISK:[DIRECTORY]ABM_AREA1.RDA;1
%RMU-I-RCLMLPAGPRC, 2008 pages processed for logical area SAMPLE_TABLE
in physical area DISK:[DIRECTORY]ABM_AREA1.RDA;1
%RMU-I-RCLMLPAGFREED, 1992 clump pages freed for logical area
SAMPLE_TABLE in physical area DISK:[DIRECTORY]ABM_AREA1.RDA;1
$
$ RMU/RECLAIM/LOG/LAREA=(SAMPLE_TABLE,SAMPLE_TABLE2)/FREE_PAGES
 ABM_SAMPLE.RDB
%RMU-I-RCLMLAREA, Reclaiming logical area SAMPLE_TABLE in physical
area DISK:[DIRECTORY]ABM_AREA1.RDA;1
%RMU-I-RCLMLPAGPRC, 2008 pages processed for logical area SAMPLE_TABLE
in physical area DISK:[DIRECTORY]ABM_AREA1.RDA;1
%RMU-I-RCLMLPAGFREED, 1992 clump pages freed for logical area
SAMPLE_TABLE in physical area DISK:[DIRECTORY]ABM_AREA1.RDA;1
%RMU-I-RCLMLAREA, Reclaiming logical area SAMPLE_TABLE2 in physical
area DISK:[DIRECTORY]ABM_AREA2.RDA;1
%RMU-I-RCLMLPAGPRC, 2008 pages processed for logical area
SAMPLE_TABLE2 in physical area DISK:[DIRECTORY]ABM_AREA2.RDA;1
%RMU-I-RCLMLPAGFREED, 1992 clump pages freed for logical area
SAMPLE_TABLE2 in physical area DISK:[DIRECTORY]ABM_AREA2.RDA;1
$
$ RMU/RECLAIM/LOG/LAREA=NDX_NAME/FREE_PAGES TEST_DATABASE.RDB
%RMU-I-RCLMLAREA, Reclaiming logical area NDX_NAME in
physical area DISK:[DIRECTORY]DB_DEFAULT.RDA;1
%RMU-I-RCLMLPAGPRC, 12 pages processed for logical area
NDX_NAME in physical area DISK:[DIRECTORY]DB_DEFAULT.RDA;1
```

```
%RMU-I-RCLMLPAGFREED, 3 clump pages freed for logical area
NDX_NAME in physical area DISK:[DIRECTORY]DB_DEFAULT.RDA;1
$
```

Examples using /LAREA and partitioned index logical area

The following example shows the Free_pages qualifier specified in an RMU Reclaim command with the Larea qualifier to free unused page clumps in an index logical area partitioned among different storage areas. Each index logical area partition is processed separately.

```
$ RMU/RECLAIM/FREE_PAGES/LOG/LAREA=INDEXA TEST_DATABASE.RDB
%RMU-I-RCLMLAREA, Reclaiming logical area INDEXA in physical area
DISK:[DIRECTORY]INDEXA_1.RDA;1
%RMU-I-RCLMLPAGPRC, 8 pages processed for logical area INDEXA in
physical area DISK:[DIRECTORY]INDEXA_1.RDA;1
%RMU-I-RCLMLPAGFREED, 3 clump pages freed for logical area INDEXA in
physical area DISK:[DIRECTORY]INDEXA_1.RDA;1
%RMU-I-RCLMLAREA, Reclaiming logical area INDEXA in physical area
DISK:[DIRECTORY]INDEXA_2.RDA;1
%RMU-I-RCLMLPAGPRC, 8 pages processed for logical area INDEXA in
physical area DISK:[DIRECTORY]INDEXA_2.RDA;1
%RMU-I-RCLMLPAGFREED, 3 clump pages freed for logical area INDEXA in
physical area DISK:[DIRECTORY]INDEXA_2.RDA;1
%RMU-I-RCLMLAREA, Reclaiming logical area INDEXA in physical area
DISK:[DIRECTORY]INDEXA_3.RDA;1
%RMU-I-RCLMLPAGPRC, 8 pages processed for logical area INDEXA in
physical area DISK:[DIRECTORY]INDEXA_3.RDA;1
%RMU-I-RCLMLPAGFREED, 3 clump pages freed for logical area INDEXA in
physical area DISK:[DIRECTORY]INDEXA_3.RDA;1
%RMU-I-RCLMLAREA, Reclaiming logical area INDEXA in physical area
DISK:[DIRECTORY]INDEXA_4.RDA;1
%RMU-I-RCLMLPAGPRC, 8 pages processed for logical area INDEXA in
physical area DISK:[DIRECTORY]INDEXA_4.RDA;1
%RMU-I-RCLMLPAGFREED, 3 clump pages freed for logical area INDEXA in
physical area DISK:[DIRECTORY]INDEXA_4.RDA;1
$
```

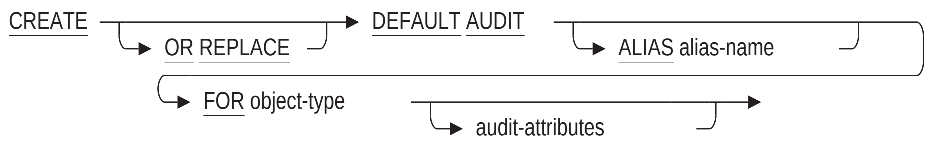
Examples showing warnings

In the following examples, warning messages are output even if /LOG is not specified in the RMU Reclaim Free_pages commands if a mixed storage area is specified or a logical area could not be processed because of a lock conflict with another user. The optional Lock_timeout qualifier is specified.

```
$ RMU/RECLAIM/LOG/AREA=DEPARTMENTS/FREE_PAGES MF_PERSONNEL.RDB
%RMU-W-RCLMMIXIGN, Mixed area DEPARTMENTS not processed if
RMU/RECLAIM/FREE_PAGES
$
$ RMU/RECLAIM/NOLOG/AREA=ABM_AREA1/FREE_PAGES/LOCK_TIMEOUT=600 -
$_ ABM_SAMPLE.RDB
%RMU-W-RCLMARNOTPRC, Area ABM_AREA1 could not be processed due to a
lock conflict
$
$ RMU/RECLAIM/LOG/LAREA=SAMPLE_TABLE/FREE_PAGES/LOCK_TIMEOUT=1200 -
$_  ABM_SAMPLE.RDB
%RMU-I-RCLMLAREA, Reclaiming logical area SAMPLE_TABLE in physical
area DISK:[DIRECTORY]ABM_AREA1.RDA;1
%RMU-W-RCLMLARNOTPRC, Logical area SAMPLE_TABLE could not be processed
due to a lock conflict
%RMU-I-RCLMLPAGPRC, 0 pages processed for logical area SAMPLE_TABLE in
physical area DISK:[DIRECTORY]ABM_AREA1.RDA;1
%RMU-I-RCLMLPAGFREED, 0 clump pages freed for logical area
SAMPLE_TABLE in physical area DISK:[DIRECTORY]ABM_AREA1.RDA;1
$
```

4.1.5 CREATE DEFAULT AUDIT Command Supports OR REPLACE Clause

This release of Oracle Rdb enhances the CREATE DEFAULT AUDIT statement by allowing the OR REPLACE clause.



Arguments

- OR REPLACE

  If the OR REPLACE clause is used and the referenced object-type exists, then it will be modified using the specified audit flags and comment. Any attributes that are not specified will assume their default values.

  Note

  Any protections granted to the object by the GRANT statement are not replaced. They would need to be removed using the REVOKE statement.

  If the referenced object-type does not exist, then it will be created as if the OR REPLACE clause was not used.

Example

This example shows the CREATE DEFAULT AUDIT statement adding a new table object (which is always named RDB$DEFAULT_AUDIT_TABLE).

```
SQL> create default audit
cont>     for table
cont>     all privileges
cont>     comment is 'Add a default audit table so we can inherit an ACL'
cont> ;
SQL>
SQL> grant select, delete, update, insert, show on rdb$default_audit_table to
testuser2;
SQL> grant select, dbctrl on rdb$default_audit_table to testuser3;
SQL> grant show on rdb$default_audit_table to public;
SQL>
SQL> --> display the attributes for the default table
SQL> --> note: that TESTUSER1 was the executor of the CREATE statement
SQL> show protection on table rdb$default_audit_table;
Protection on Table RDB$DEFAULT_AUDIT_TABLE
    (IDENTIFIER=[TEST,TESTUSER3],ACCESS=SELECT+DBCTRL)
    (IDENTIFIER=[TEST,TESTUSER2],ACCESS=SELECT+INSERT+UPDATE+DELETE+SHOW)
    (IDENTIFIER=[TEST,TESTUSER1],ACCESS=SELECT+INSERT+UPDATE+DELETE+SHOW+
 CREATE+ALTER+DROP+DBCTRL+REFERENCES)
    (IDENTIFIER=[*,*],ACCESS=SHOW)
SQL> show audit on table rdb$default_audit_table;
Audit information for Table RDB$DEFAULT_AUDIT_TABLE
 Audit Privileges:
    ALL
 Alarm Privileges:
    ALL

SQL> show table (comment) rdb$default_audit_table;
Information for table RDB$DEFAULT_AUDIT_TABLE
```

```
Comment on table RDB$DEFAULT_AUDIT_TABLE:
Add a default audit table so we can inherit an ACL

A global temporary table.
On commit Delete rows

SQL>
```

At some later time, the table template object can be created or replaced using the CREATE OR REPLACE DEFAULT AUDIT statement.

```
SQL> create or replace default audit
cont>     for table
cont>     type is (audit)
cont>     privileges (success, failure)
cont>     comment is 'Only audit SUCCESS and FAILURE'
cont> ;
SQL>
SQL> --> show that the protections are retained by OR REPLACE
SQL> show protection on table rdb$default_audit_table;
Protection on Table RDB$DEFAULT_AUDIT_TABLE
    (IDENTIFIER=[TEST,TESTUSER3],ACCESS=SELECT+DBCTRL)
    (IDENTIFIER=[TEST,TESTUSER2],ACCESS=SELECT+INSERT+UPDATE+DELETE+SHOW)
    (IDENTIFIER=[TEST,TESTUSER1],ACCESS=SELECT+INSERT+UPDATE+DELETE+SHOW
 +CREATE+ALTER+DROP+DBCTRL+REFERENCES)
    (IDENTIFIER=[*,*],ACCESS=SHOW)
SQL> show audit on table rdb$default_audit_table;
Audit information for Table RDB$DEFAULT_AUDIT_TABLE
 Audit Privileges:
    SUCCESS,FAILURE

SQL> show table (comment) rdb$default_audit_table;
Information for table RDB$DEFAULT_AUDIT_TABLE

Comment on table RDB$DEFAULT_AUDIT_TABLE:
Only audit SUCCESS and FAILURE

A global temporary table.
On commit Delete rows

SQL>
SQL> commit;
SQL>
```

4.1.6 System Privileges Feature

This release of Oracle Rdb introduces Database System Privileges - an enhancement for database security.

Introduction

A typical Oracle Rdb database will have a security policy defined by granting privileges to users of the database and, when objects are created (tables, sequences, and so on), granting access to those objects. In addition, roles (also known as rights identifiers) may be granted specific access and any user assigned that role may inherit its access rights.

These granted privileges are stored in an access control list (ACL) with an entry for a user or role known as an access control entry (ACE).

Please refer to the SQL Reference Manual GRANT statement and REVOKE statement sections for more detailed descriptions and examples.

Security Definitions

All objects are within the DATABASE security domain.

The primary security objects are: ASSERTION (an assertion is a form of constraint defined independently of a table definition. This feature is currently not available in SQL but maps to the standalone RDO constraints), CATALOG, COLLATING SEQUENCE, DOMAIN, OUTLINE, PROCEDURE, PROFILE, TABLE, ROLE, SCHEMA, SEQUENCE, SYNONYM, and USER.

For the purposes of system privileges, Oracle Rdb treats modules, functions and procedures as a single class - namely PROCEDURE.

Some primary security objects have an associated ACL that protects that object and controls access to sub-objects. TABLE includes the following sub-objects: VIEW, STORAGE MAP, and TRIGGER.

When TABLE, SEQUENCE, and PROCEDURE objects are created, they are implicitly given an ACL that grants ALL PRIVILEGES to the creator and NO PRIVILEGES to the PUBLIC (also known as [*,*]). The database administrator can override this default ACL for new primary objects:

1. For tables and views, the PUBLIC access control entry will be inherited from the database access control entry for the DEFAULT user. Note: not all OpenVMS systems have a DEFAULT user defined in the system user authorization file (UAF) so that would be created by the system manager if required.

   ```
   SQL> grant show
   cont>     on database alias paysys
   cont>     to default
   cont> ;
   SQL>
      .
      .
      .
   SQL> create table paysys.CONTROL_TABLE
   cont>     (identifier_value integer generated by default as identity
   cont>     );
   SQL>
   SQL> show protection on table paysys.CONTROL_TABLE;
   Protection on Table PAYSYS.CONTROL_TABLE
       (IDENTIFIER=[ADMIN,DATABASE],ACCESS=SELECT+INSERT+UPDATE+DELETE+
        SHOW+CREATE+ALTER+DROP+DBCTRL+REFERENCES)
       (IDENTIFIER=[*,*],ACCESS=SHOW)
   SQL>
   ```

2. Alternately, the CREATE DEFAULT AUDIT statement can be used to define template security objects in the database. These special objects are used for audit and protection inheritance. When an object (table, sequence, etc) is created, then the access control list from the security template is inherited. Note that choice (1) above will also be applied to the TABLE and VIEW default audit templates.

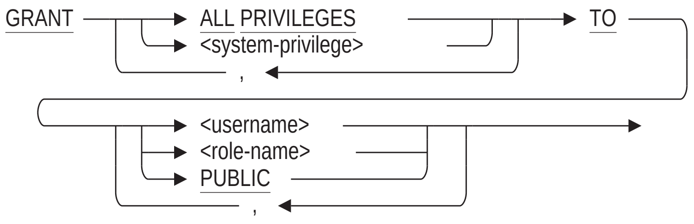
   This example shows the **default audit** object creation and then being granted a default access for PUBLIC and specific access for users and roles.

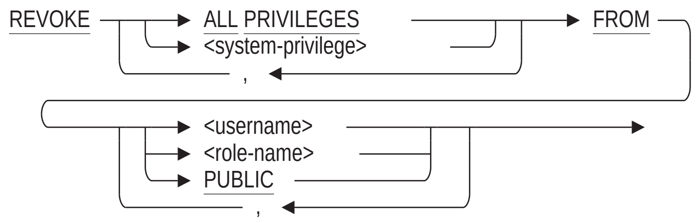
```
SQL> create default audit
cont>     alias PAYSYS
cont>     for table
cont> ;
SQL>
SQL> grant select
cont>     on table PAYSYS.rdb$default_audit_table
cont>     to m_smith, b_lee, s_jain
cont> ;
SQL>
SQL> grant select, insert, delete, update
cont>     on table PAYSYS.rdb$default_audit_table
cont>     to paysys_admin
cont> ;
SQL>
SQL> grant show
cont>     on table PAYSYS.rdb$default_audit_table
cont>     to PUBLIC
cont> ;
SQL>
   .
   .
   .
SQL> create table paysys.CONTROL_TABLE
cont>     (identifier_value integer generated by default as identity
cont>     );
SQL>
SQL> show protection on table paysys.CONTROL_TABLE;
Protection on Table PAYSYS.CONTROL_TABLE
    (IDENTIFIER=PAYSYS_ADMIN,ACCESS=SELECT+INSERT+UPDATE+DELETE)
    (IDENTIFIER=[RDB,S_JAIN],ACCESS=SELECT)
    (IDENTIFIER=[DEV,B_LEE],ACCESS=SELECT)
    (IDENTIFIER=[AUDITOR,M_SMITH],ACCESS=SELECT)
    (IDENTIFIER=[ADMIN,DATABASE],ACCESS=SELECT+INSERT+UPDATE+DELETE+
     SHOW+CREATE+ALTER+DROP+DBCTRL+REFERENCES)
    (IDENTIFIER=[*,*],ACCESS=SHOW)
SQL>
```

Database Vault Feature

The security policy implemented through ACLs can be overridden at runtime by a suitably privileged OpenVMS user. That is, an OpenVMS power user might be able to attach and select data from a table even if they do not have database or table access granted by an access control entry. This override ability can be limited on a per database level by enabling the DATABASE VAULT attribute using ALTER DATABASE ... DATABASE VAULT IS ENABLED. When DATABASE VAULT is enabled, only the access control lists and database system privileges are used to determine access to database objects. This can prevent accidental override of the security policy. See the SQL Reference Manual DATABASE VAULT Appendix for further details.

Syntax





Arguments

- ALL PRIVILEGES

  All privileges will be granted (or revoked) from the listed users and roles.

- system-privilege

  Refer to Table 4–1, System Privileges for a list of supported system privileges.

- TO username
  TO role-name
  TO PUBLIC

  Specifies the user name, role name, or the PUBLIC user to which you want to grant the system privilege. The PUBLIC user is the user name associated with all anonymous users who access the database.

  Note

  Oracle recommends that you only grant system privileges to trusted users. If system privileges are granted to roles then only assign those roles to trusted users.

  If the database is defined as SECURITY CHECK IS INTERNAL and the user or role name exists as an operating system user or rights identifier, Oracle Rdb will automatically create the user or role name when you issue the GRANT statement.

- FROM username
  FROM role-name
  FROM PUBLIC

  Specifies the user, role, or the PUBLIC user from which the specified role is to be revoked.

Database System Privileges

System privileges are associated with specific users (CREATE USER) and roles (CREATE ROLE) within the database. They are assigned by a user with SECURITY privilege on the database by the GRANT statement and removed by the REVOKE statement.

For example,

```
SQL> create user J_JONES identified externally;
SQL> create role DB_PROGRAMMER identified externally;
SQL>
SQL> grant create any sequence, create any procedure,
cont> create any temporary table to DB_PROGRAMMER;
SQL> grant create any trigger, create any index to J_JONES;
```

In this example, the database uses the default SECURITY CHECKING IS EXTERNAL therefore the granting of the rights identifier DB_PROGRAMMER must be performed by OpenVMS. Use the AUTHORIZE utility to grant the rights identifier to specific users.

```
$ run sys$system:authorize
UAF> GRANT/ID DB_PROGRAMMER J_JONES
UAF>
```

When J_JONES attaches to the database, they will receive the benefits of their own granted system privileges (if any) as well as those granted to the assigned role (DB_PROGRAMMER). This is true even for DATABASE VAULT protected databases.

If a database is defined as SECURITY CHECKING IS INTERNAL, then the GRANT statement is used to associate roles created by CREATE ROLE with specific users.

```
SQL> grant DB_PROGRAMMER to J_JONES;
SQL>
```

Refer to the SQL Reference Manual GRANT Statement: Roles section for more details and examples.

Implied System Privileges

In prior releases of Oracle Rdb, there were implied system privileges based on the database level CREATE, ALTER, DROP and SECURITY privileges that may have been granted to the user via the database ACL.

CREATE implies the permission to CREATE ANY object (except PROFILE, ROLE, and USER), and ALTER implies the permission to ALTER any object (except PROFILE, ROLE, and USER), DROP implies the permission to DROP ANY object (except PROFILE, ROLE, and USER), and SECURITY implies the permission to CREATE, ALTER, and DROP ANY PROFILE, ROLE or USER.

These implied system privileges can be displayed when attaching to a database and performing a SHOW PRIVILEGE ON DATABASE command. The implied system privileges (if any) are displayed. This is similar to the access privileges shown by this command. They reflect the ACL on the database as well as inherited access based on special privileges such as DBADM and OpenVMS privileges.

```
SQL> show privileges on database rdb$dbhandle;
Privileges on Alias RDB$DBHANDLE
    (IDENTIFIER=[RDB,RDBUSER2],ACCESS=SELECT+CREATE+ALTER+DROP)

Current system privileges:
     Granted Create Any
     COLLATING SEQUENCE, DOMAIN, ASSERTION, SESSION, OUTLINE, PROCEDURE,
     TABLE, SEQUENCE, VIEW
     Granted Alter Any
     COLLATING SEQUENCE, DOMAIN, ASSERTION, DATABASE, OUTLINE
     Granted Drop Any
     COLLATING SEQUENCE, DOMAIN, ASSERTION, DATABASE, OUTLINE
SQL>
```

Note that ALTER and DROP for primary database objects are not implicitly inherited from the database ALTER and DROP database privileges. This is because these operations are controlled by the object's own access control list.

Fine Tuning

Although system privileges are now available in Oracle Rdb, the database administrator is not required to use them. The database will operate as it did in previous releases.

Making use of this enhanced privilege system requires that the database administrator perform these tasks.

- Use CREATE USER for each database user to which system privileges need to be granted.
- Use CREATE ROLE for any rights identifiers used to fine tune access.

Not all users and roles that access the database need to be created in the database as Rdb will still use them as in prior releases to select matching access control entries. However, users and roles must be created to allow the database administrator to grant system privileges as these objects are used to store the current privilege set.

- Use the GRANT statement to manage the CREATE ANY, ALTER ANY, DROP ANY, and TRUNCATE ANY privileges and assign them to users and roles.
- Use REVOKE on the DATABASE ALIAS to remove the CREATE, ALTER, DROP or SECURITY privileges that were previously granted to those uses. This step is required so that those database level privileges do not interfere with the fine control of system privileges.

This change will limit those users according to the new security policy. The SHOW USER and SHOW ROLE statements will display all the granted system privileges. The SHOW PRIVILEGES ON DATABASE will show the augmented system privileges based on:

- Database ACL entries,
- Granted system privileges for the attaching user,
- Granted system privileges for all rights (roles) assigned to the user,
- OpenVMS privileges (when DATABASE VAULT is enabled there will be none used)

All access control is established at ATTACH time so changes made to the USER, ROLE or DATABASE access will not have an effect until the next database ATTACH.

Note

Oracle recommends that you only grant system privileges to trusted users. If system privileges are granted to roles then only assign those roles to trusted users.

Table 4–1 System Privileges

Operation Type	Object Type	Description
ALL PRIVILEGES		Can be used to GRANT or REVOKE all system privileges to a USER or ROLE.
ALTER ANY ...		
	ASSERTION	Permits the holder to execute the RDO CHANGE CONSTRAINT command and the SQL ALTER CONSTRAINT and COMMENT ON CONSTRAINT statements.
	CATALOG	Permits the holder to execute the ALTER CATALOG statement. The database must have multischema enabled; ALTER DATABASE ... MULTISCHEMA IS ON;
	COLLATING SEQUENCE	Permits the holder to execute the ALTER COLLATING SEQUENCE statement.
	DOMAIN	Permits the holder to execute the ALTER DOMAIN statement.
	DATABASE	Permits the holder to execute the ALTER DATABASE and COMMENT ON DATABASE statements.
	INDEX	Permits the holder to execute the ALTER INDEX statement.
	OUTLINE	Permits the holder to execute the ALTER OUTLINE statement.
	PROCEDURE	Permits the holder to execute the ALTER FUNCTION, ALTER MODULE and ALTER PROCEDURE statements.
	PROFILE	Permits the holder to execute the ALTER PROFILE and ALTER DEFAULT PROFILE statements. When the profile exists, the holder can also execute the CREATE OR REPLACE PROFILE and CREATE OR REPLACE DEFAULT PROFILE statements.
	ROLE	Permits the holder to execute the ALTER ROLE statement.
	SEQUENCE	Permits the holder to execute the ALTER SEQUENCE statement. When the sequence exists, the holder can also execute the CREATE OR REPLACE SEQUENCE statement.
	SCHEMA	Permits the holder to execute the ALTER SCHEMA statement. The database must have multischema enabled: ALTER DATABASE ... MULTISCHEMA IS ON;
	STORAGE MAP	Permits the holder to execute the ALTER STORAGE MAP statement.

(continued on next page)

Table 4–1 (Cont.) System Privileges

Operation Type	Object Type	Description
	SYNONYM	Permits the holder to execute the ALTER SYNONYM statement. When the synonym exists, the holder can also execute the CREATE OR REPLACE SYNONYM statements. The database must have synonyms enabled: ALTER DATABASE ... SYNONYMS ARE ENABLED;
	TABLE	Permits the holder to execute the ALTER TABLE statement.
	TEMPORARY TABLE	Permits the holder to execute the ALTER TEMPORARY TABLE statement or CREATE INFORMATION TABLE statement when CREATE ANY TABLE privilege is not granted.
	TRIGGER	Permits the holder to execute the ALTER TRIGGER statement.
	USER	Permits the holder to execute the ALTER USER statement.
	VIEW	Permits the holder to execute the ALTER VIEW statement. When the view exists, the holder can also execute the CREATE OR REPLACE VIEW statement.
CREATE ...		
	SESSION	Permits the holder to execute ATTACH, CONNECT, DECLARE ALIAS, SET SESSION AUTHORIZATION, and other session starting statements.
CREATE ANY ...		
	ASSERTION	Permits the holder to execute the RDO DEFINE CONSTRAINT command. The SQL equivalent to DEFINE CONSTRAINT would be a table level constraint. Such definitions are managed by TABLE privileges, therefore this privilege does not apply to SQL.
	CATALOG	Permits the holder to execute the CREATE CATALOG statement. The database must have multischema enabled: ALTER DATABASE ... MULTISCHEMA IS ON;
	COLLATING SEQUENCE	Permits the holder to execute the CREATE COLLATING SEQUENCE statement.
	DOMAIN	Permits the holder to execute the CREATE DOMAIN statement.
	INDEX	Permits the holder to execute the CREATE INDEX statement.
	OUTLINE	Permits the holder to execute the CREATE OUTLINE statement.

(continued on next page)

Table 4–1 (Cont.) System Privileges

Operation Type	Object Type	Description
	PROCEDURE	Permits the holder to execute the CREATE FUNCTION, CREATE MODULE and CREATE PROCEDURE statements.
	PROFILE	Permits the holder to execute the CREATE PROFILE and CREATE DEFAULT PROFILE statements.
	ROLE	Permits the holder to execute the CREATE ROLE statement.
	SEQUENCE	Permits the holder to execute the CREATE SEQUENCE statement.
	SCHEMA	Permits the holder to execute the CREATE SCHEMA statement. The database must have multischema enabled: ALTER DATABASE ... MULTISCHEMA IS ON;
	STORAGE MAP	Permits the holder to execute the CREATE STORAGE MAP statement.
	SYNONYM	Permits the holder to execute the CREATE SYNONYM statement. The database must have synonyms enabled: ALTER DATABASE ... SYNONYMS ARE ENABLED;
	TABLE	Permits the holder to execute the CREATE TABLE statement.
	TEMPORARY TABLE	Permits the holder to execute the CREATE TEMPORARY TABLE and CREATE INFORMATION TABLE statements when CREATE ANY TABLE privilege is not granted.
	TRIGGER	Permits the holder to execute the CREATE TRIGGER statement.
	USER	Permits the holder to execute the CREATE USER statement.
	VIEW	Permits the holder to execute the CREATE VIEW statement.
DROP ANY ...		
	ASSERTION	Permits the holder to execute the RDO DELETE CONSTRAINT command or the SQL DROP CONSTRAINT statement.
	CATALOG	Permits the holder to execute the DROP CATALOG statement. The database must have multischema enabled: ALTER DATABASE ... MULTISCHEMA IS ON;
	COLLATING SEQUENCE	Permits the holder to execute the DROP COLLATING SEQUENCE statement.
	DOMAIN	Permits the holder to execute the DROP DOMAIN statement.
	DATABASE	Permits the holder to execute the DROP DATABASE statement.

(continued on next page)

Table 4–1 (Cont.) System Privileges

Operation Type	Object Type	Description
	INDEX	Permits the holder to execute the DROP INDEX statement.
	OUTLINE	Permits the holder to execute the DROP OUTLINE statement.
	PROCEDURE	Permits the holder to execute the DROP FUNCTION, DROP MODULE and DROP PROCEDURE statements.
	PROFILE	Permits the holder to execute the DROP PROFILE and DROP DEFAULT PROFILE statements.
	ROLE	Permits the holder to execute the DROP ROLE statement.
	SEQUENCE	Permits the holder to execute the DROP SEQUENCE statement.
	SCHEMA	Permits the holder to execute the DROP SCHEMA statement. The database must have multischema enabled: ALTER DATABASE ... MULTISCHEMA IS ON;
	STORAGE MAP	Permits the holder to execute the DROP STORAGE MAP statement.
	SYNONYM	Permits the holder to execute the DROP SYNONYM statement. The database must have synonyms enabled: ALTER DATABASE ... SYNONYMS ARE ENABLED;
	TABLE	Permits the holder to execute the DROP TABLE statement.
	TEMPORARY TABLE	Permits the holder to execute the DROP TEMPORARY TABLE statement or DROP INFORMATION TABLE statement when DROP ANY TABLE privilege is not granted.
	TRIGGER	Permits the holder to execute the DROP TRIGGER statement.
	USER	Permits the holder to execute the DROP USER statement.
	VIEW	Permits the holder to execute the DROP VIEW statement.
TRUNCATE ...		
	ANY TABLE	Permits the holder to execute the TRUNCATE TABLE statement. This privilege effectively allows the user to temporarily disable triggers during the TRUNCATE operation and assume DELETE access to the table.
	TABLE	This privilege is similar to the TRUNCATE ANY TABLE privilege but requires that the user also be granted DELETE access to the table. Permits the user to execute the TRUNCATE TABLE statement without further privilege checking for BEFORE and AFTER DELETE triggers.

Usage Notes

- You must have the SECURITY privilege on the database to grant a system privilege to a user or a role.
- You must have the SECURITY privilege on the database to revoke a system privilege from a user or a role.
- The TEMPORARY TABLE class of privileges is considered a subset of TABLE and allows the database administrator to grant privileges to a user but only for logical tables not physical (base) tables.
- If the user has CREATE ANY TABLE, then the CREATE ANY TEMPORARY TABLE privilege is not required. Similarly, ALTER ANY TEMPORARY TABLE and DROP ANY TEMPORARY TABLE are not used if ALTER ANY TABLE or DROP ANY TABLE is granted.
- The SHOW PRIVILEGES ON DATABASE statement displays the current active system privileges. This is based on the current user system privileges, system privileges inherited from granted roles, inherited privileges from the database access control list and OpenVMS process privileges.

  It does not display any privileges for features which are not enabled for the database. CATALOG and SCHEMA privileges will not be displayed if MULTISCHEMA is not enabled. SYNONYM privileges will not be displayed if SYNONYMS are not enabled. However, the SHOW USER and SHOW ROLE statements will display all granted privileges even if that privilege has no application in the current database configuration.

4.1.7 Database Vault Feature

This release of Oracle Rdb introduces the DATABASE VAULT functionality.

The goal of DATABASE VAULT is to avoid accidental database access by an OpenVMS privileged user when the database security policy (ACL) should prevent such access.

This feature allows the database administrator to enforce an access policy for all attached database users by disabling the use of OpenVMS privileges as overrides to the database access control list.

DATABASE VAULT can be enabled by any of these commands.

- The SQL CREATE DATABASE ... DATABASE VAULT IS ENABLED statement.
- The SQL ALTER DATABASE ... DATABASE VAULT IS ENABLED statement.
- The SQL IMPORT DATABASE ... DATABASE VAULT IS ENABLED statement.

Note

If a database with DATABASE VAULT enabled is exported, then an IMPORT DATABASE will implicitly execute the DATABASE VAULT IS ENABLED action without that clause being required on the statement.

- RMU/SET DATABASE/DATABASE_VAULT=ENABLED command

DATABASE VAULT can be disabled by any of these commands.

- The SQL ALTER DATABASE ... DATABASE VAULT IS DISABLED statement.
- The SQL IMPORT DATABASE ... DATABASE VAULT IS DISABLED statement.
- RMU/SET DATABASE/DATABASE_VAULT=DISABLED command

Before executing any of these commands, the user must be granted (at least temporarily) the rights identifier RDBVMS$DATABASE_VAULT_MANAGER that is added to the system during installation.

For example, the SET RIGHTS_LIST DCL command can be used to temporarily enable it.

```
$ SET RIGHTS_LIST /ENABLE RDBVMS$DATABASE_VAULT_MANAGER
$
$ rmu/set database mf_personnel/database_vault=enable
%RMU-I-MODIFIED, Database state modified
%RMU-W-DOFULLBCK, full database backup should be done to ensure future
recovery
$
$ SET RIGHTS_LIST /DISABLE RDBVMS$DATABASE_VAULT_MANAGER
```

Please refer to the Oracle Rdb SQL Reference Manual, Appendix J for more details. This includes a description of the new DBVAULT audit class that can be used to audit changes to the DATABASE VAULT settings of a database.

4.1.8 SET FLAGS Keyword for Hash Join Feature - HASHING

This release of Oracle Rdb has added a flag to control the HASH JOIN feature of the Rdb optimizer. The HASHING flag can be used with the SET FLAGS statement or the RDMS$SET_FLAGS logical name to enable this feature. When enabled, the optimizer will attempt to engage the HASH JOIN feature during query solution. The default is NOHASHING(JOINS).

The following example shows the HASHING flag in use.

```
SQL> set flags 'strategy,detail(2)';
SQL> set flags 'hashing(joins)';
SQL>
SQL> select e.employee_id, e.birthday, jh.job_start
cont> from employees e, job_history jh
cont> where e.employee_id = jh.employee_id
cont>  and jh.job_end is null
cont> ;
Tables:
  0 = EMPLOYEES
  1 = JOB_HISTORY
Conjunct: 0.EMPLOYEE_ID = 1.EMPLOYEE_ID
Hash Q1
  Outer Build
  Match_Key:0.EMPLOYEE_ID
    Get     Retrieval by index of relation 0:EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:0]
  Inner Probe
  Match_Key:1.EMPLOYEE_ID
    Conjunct: MISSING (1.JOB_END)
    Get     Retrieval by index of relation 1:JOB_HISTORY
      Index name  JH_EMPLOYEE_ID [0:0]
Table=0:EMPLOYEES #Buckets=131 #Hits=61 #Collisions=39 #Dups=0 #Dups_Chain=0
Load_Factor=   4.656488549618321E-001
 E.EMPLOYEE_ID   E.BIRTHDAY    JH.JOB_START
 00164           28-Mar-1947   21-Sep-1981
 00165           15-May-1954    8-Mar-1981
.
.
.
100 rows selected
SQL>
```

To disable the flag, use 'NOHASHING(JOINS)'. The setting is displayed by the SHOW FLAGS statement.

See also the new logical name RDMS$ENABLE_HASH_JOIN. Defining this logical name to true ("T", "t", "Y", "y" or "1") instructs the Rdb optimizer to try to use in-memory HASH JOIN to solve queries.

4.1.9 JOIN BY HASH Clause in CREATE OUTLINE Statement

This release of Oracle Rdb adds a new JOIN BY HASH clause to the CREATE OUTLINE statement and the OPTIMIZE OUTLINE clause of the select statement.

The following example shows the new syntax and the resulting query strategy.

```
SQL> create outline QO_1
cont> id '352E2736F133A6A322A3C935DB2CBE12'
cont> mode 0
cont> as (
cont>   query (
cont>     subquery (
cont>       EMPLOYEES 0 access path index EMP_EMPLOYEE_ID
cont>         join by hash to
cont>       SALARY_HISTORY 1 access path index SH_EMPLOYEE_ID
cont>       )
cont>     )
cont>   )
cont> compliance optional;
SQL>
SQL> set flags 'strategy,detail(2)';
SQL>
SQL> select
cont>     e.employee_id, sh.salary_start, sh.salary_amount
cont> from
cont>     employees e
cont>     inner join
cont>     salary_history sh on (e.employee_id = sh.employee_id
cont>                          and sh.salary_end is null)
cont> optimize using QO_1
cont> ;
~S: Outline "QO_1" used
Tables:
  0 = EMPLOYEES
  1 = SALARY_HISTORY
Conjunct: 0.EMPLOYEE_ID = 1.EMPLOYEE_ID
Hash Q1
  Outer Build
  Match_Key:0.EMPLOYEE_ID
    Index only retrieval of relation 0:EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:0]
  Inner Probe
  Match_Key:1.EMPLOYEE_ID
    Conjunct: MISSING (1.SALARY_END)
    Get     Retrieval by index of relation 1:SALARY_HISTORY
      Index name  SH_EMPLOYEE_ID [0:0]
Table=0:EMPLOYEES #Buckets=131 #Hits=61 #Collisions=39 #Dups=0 #Dups_Chain=0
Load_Factor=   4.656488549618321E-001
```

```
 E.EMPLOYEE_ID   SH.SALARY_START   SH.SALARY_AMOUNT
 00164           14-Jan-1983             $51,712.00
 00165            1-Jul-1982             $11,676.00
.
.
.
```

Note

This syntax, JOIN BY HASH, cannot be applied remotely to an older version of Oracle Rdb. An error such as this will be returned.

```
SQL> create outline QO_1
cont> id '352E2736F133A6A322A3C935DB2CBE12'
cont> mode 0
cont> as (
cont>   query (
cont>     subquery (
cont>       EMPLOYEES 0 access path index EMP_EMPLOYEE_ID
cont>         join by hash to
cont>       SALARY_HISTORY 1 access path index SH_EMPLOYEE_ID
cont>       )
cont>     )
cont>   )
cont> compliance optional;
%SQL-F-UNSUPVER, Operation is unsupported for version of database
-SQL-F-UNSUPFEATURE, feature JOIN BY HASH is not supported
```

4.1.10 Hash Join Feature

Status: BETA

This release of Oracle Rdb includes a new optimization method known as Hash Join. A Hash Join is performed by hashing (mapping) one set of data into virtual memory based on the join columns and reading the other table to probe into the hash table to locate matching rows.

Typically, a Hash Join has a lower cost compared to the alternate of sorting when the hash table can be held entirely in memory, with the total cost amounting to very little other than the cost of reading the data sets. The cost rises if the hash table has to be spilled over to a temporary file.

Hash Join is only used for equi-joins. In general, Hash Join is a better solution for joining large numbers of rows in an equi-join.

Applications that in the past used Match Join might (unknowingly) rely on the implicit use of SORT during the query solution. However, as no implicit SORT is performed, the data might appear in a different order with Hash Join. Adding an ORDER BY will sort the result data but no longer sort the inputs to the join.

Note

Not all queries will be solved using HASH JOIN if this optional feature is enabled. Use the `SET FLAGS 'STRATEGY,DETAIL(3)'` to see a report of the current restrictions which cause the HASH JOIN method to be rejected.

Enabling HASH JOIN

The optimizer, by default, does not try the HASH JOIN method in solutions. This action must be enabled by the programmer in one of the following ways. The optimizer will then include HASH JOIN as part of its solution matching, but it may be rejected for various reasons.

- Define the logical name RDMS$ENABLE_HASH_JOIN.

  Defining this logical name to true ("T", "t", "Y", "y" or "1") instructs the Rdb optimizer to try to use in-memory HASH JOIN to solve queries.
- Defining the logical name RDMS$SET_FLAGS or using the SET FLAGS statement with the string 'HASHING(JOINS)'. See Section 4.1.8 for more details.
- The new JOIN BY HASH clause in CREATE OUTLINE statement and the OPTIMIZE OUTLINE clause of the select statement. See Section 4.1.9 for more details.
- Specifying the OPTIMIZE FOR HASH JOIN on the select statement.

```
SQL> select e.employee_id, (sh.salary_end - sh.salary_start) month (3)
cont> from employees e, salary_history sh
cont> where e.employee_id = sh.employee_id
cont>       and e.employee_id <= '00164'
cont>       and sh.salary_end is not null
cont> optimize for hash join
cont> ;
.
.
.
```

Example

This example shows the strategy used by the optimizer when HASH JOIN is enabled.

```
SQL> set flags 'STRATEGY,DETAIL(2)';
SQL> set flags 'HASHING(JOINS)';
SQL>
SQL> select e.employee_id, (sh.salary_end - sh.salary_start) month (3)
cont> from employees e, salary_history sh
cont> where e.employee_id = sh.employee_id
cont>       and e.employee_id <= '00164'
cont>       and sh.salary_end is not null
cont> ;
Tables:
  0 = EMPLOYEES
  1 = SALARY_HISTORY
Conjunct: 0.EMPLOYEE_ID = 1.EMPLOYEE_ID
Hash Q1
  Outer Build
  Match_Key:0.EMPLOYEE_ID
    Index only retrieval of relation 0:EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:1]
        Keys: 0.EMPLOYEE_ID <= '00164'
  Inner Probe
  Match_Key:1.EMPLOYEE_ID
    Conjunct: NOT MISSING (1.SALARY_END)
    Conjunct: 1.EMPLOYEE_ID <= '00164'
    Get     Retrieval by index of relation 1:SALARY_HISTORY
      Index name  SH_EMPLOYEE_ID [0:1]
        Keys: 1.EMPLOYEE_ID <= '00164'
Table=0:EMPLOYEES #Buckets=47 #Hits=1 #Collisions=0 #Dups=0 #Dups_Chain=0
```

```
Load_Factor=   2.127659574468085E-002
 E.EMPLOYEE_ID
 00164            008
 00164            006
 00164            016
3 rows selected
SQL>
```

Now try the same query with NOHASHING.

```
SQL> set flags 'NOHASHING(JOINS)';
SQL>
SQL> select e.employee_id, (sh.salary_end - sh.salary_start) month (3)
cont> from employees e, salary_history sh
cont> where e.employee_id = sh.employee_id
cont>       and e.employee_id <= '00164'
cont>       and sh.salary_end is not null
cont> ;
Tables:
  0 = EMPLOYEES
  1 = SALARY_HISTORY
Conjunct: 0.EMPLOYEE_ID = 1.EMPLOYEE_ID
Match  Q1
  Outer loop      (zig-zag)
  Match_Key:0.EMPLOYEE_ID
  Index_Key:EMPLOYEE_ID
    Index only retrieval of relation 0:EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:1]
        Keys: 0.EMPLOYEE_ID <= '00164'
  Inner loop      (zig-zag)
  Match_Key:1.EMPLOYEE_ID
  Index_Key:EMPLOYEE_ID
    Conjunct: NOT MISSING (1.SALARY_END)
    Conjunct: 1.EMPLOYEE_ID <= '00164'
    Get     Retrieval by index of relation 1:SALARY_HISTORY
      Index name  SH_EMPLOYEE_ID [0:1]
        Keys: 1.EMPLOYEE_ID <= '00164'
 E.EMPLOYEE_ID
 00164            008
 00164            006
 00164            016
3 rows selected
SQL>
```

4.1.11 New ALTER DATABASE ... LOAD ACL IDENTIFIERS Clause

In this release of Oracle Rdb, the database administrator can automatically and simply create users and roles in the database. This clause, LOAD ACL IDENTIFIERS, is part of the ALTER DATABASE statement and can be run as often as necessary to add new users and roles derived from the existing access control lists (ACLs) granted to the database and database objects.

The following example shows this clause on a sample database:

```
SQL> attach 'filename WAREHOUSE';
SQL>
SQL> show roles;
Roles in database with filename WAREHOUSE
 No roles found
SQL> show users;
Users in database with filename WAREHOUSE
 No users found
SQL>
SQL> disconnect all;
SQL>
SQL> alter database
cont>     filename WAREHOUSE
cont>
cont>     load acl identifiers
cont> ;
SQL>
SQL> attach 'filename WAREHOUSE';
SQL>
SQL> show roles;
Roles in database with filename WAREHOUSE
     CDD$EXTENDER
     CDD$SYSTEM
     STORES_CUST_READABLE
     STORES_EXTRACT_TEXT
     STORES_MAIL_TEXT
     STORES_PRINT_TEXT
     STORES_USER
SQL> show users;
Users in database with filename WAREHOUSE
     FLEE
     ISMITH
     JJONES
     KSTJOHN
     WH_QUERY_1
     WH_QUERY_2
     WH_QUERY_3
     WH_QUERY_4
     WH_QUERY_5
SQL>
SQL> disconnect all;
SQL>
```

When the database administrator uses the GRANT statement to give access to users and OpenVMS rights identifiers (aka roles), they are recorded in the access control lists for each object; database, table, view, column, sequence, module, and routine. This clause of the ALTER DATABASE statement reads every ACL in the database and creates USER and ROLE definitions if necessary.

Note

Some access control entries (ACEs) may use OpenVMS group identifiers (PUBLIC, [*,*], [ADMIN,*], [*], etc), or special modifier rights identifiers (BATCH, DIALUP, INTERACTIVE, LOCAL, NETWORK, REMOTE) which are not valid users and roles - these will be ignored by the LOAD ACL IDENTIFIERS clause.

In addition to the DBADM privilege required to use ALTER DATABASE, this clause also requires SECURITY on the database. Alternately, the user must be granted the ALTER ANY DATABASE, CREATE ANY USER and CREATE ANY ROLE database system privilege.

4.1.12 ALTER TABLE Statement Allows Change to READ ONLY Table

This release of Oracle Rdb adds the ability to change a table to READ ONLY access. Once committed, no other application or interactive SQL session may modify rows in the table. You can issue database definition statements (DDL) as long as they do not modify any table data. Operations on indices associated with the table are allowed when the table is in READ ONLY mode. To revert to a read-write table, the clause READ WRITE can be applied.

While the table is READ ONLY the following restrictions apply:

- The data manipulation statements INSERT, UPDATE, DELETE may not modify rows in the table.
- Table updates via LIST cursor may fail during the OPEN or CLOSE statement depending on the cursor declaration.
- The SELECT statement using the FOR UPDATE clause may fail because it tries to apply UPDATE semantics.

  ```
  SQL> select * from SAMPLE_TABLE for update;
  %RDB-E-READ_ONLY_REL, relation SAMPLE_TABLE was reserved for read access;
  updates not allowed
  ```
- The TRUNCATE TABLE statement is not permitted to truncate rows from the table.
- ALTER TABLE ... ADD COLUMN is permitted unless a DEFAULT is added for the new column (either explicitly or implicitly from a domain reference). In this case, Rdb would normally execute an UPDATE statement to include the default into each pre-existing row.
- Most other database definition statements are permitted. For instance, CREATE INDEX, ALTER INDEX ... REBUILD ALL PARTITIONS, DROP INDEX, can all be performed while the table is in this state.

The following example shows the diagnostic reported by Oracle Rdb.

```
SQL> alter table SAMPLE_TABLE
cont>    read only
cont> ;
SQL>
SQL> show table (comment) SAMPLE_TABLE;
Information for table SAMPLE_TABLE

Comment on table SAMPLE_TABLE:
Samples table

Table is set READ ONLY

SQL> truncate table SAMPLE_TABLE;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-READ_ONLY_REL, relation SAMPLE_TABLE was reserved for read access;
updates not allowed
SQL> commit;
SQL>
```

4.1.13 NULLS FIRST and NULLS LAST Options for ORDER BY Clause

This release of Oracle Rdb adds the NULLS FIRST and NULLS LAST options to the ORDER BY clause. These options control the ordering of the NULL values relative to the ordering of the key data. This is demonstrated by the four examples shown below.

```
SQL> select employee_id, salary_start, salary_end, salary_amount
cont>  from salary_history sh
cont>  where employee_id = '00164'
cont>  order by salary_end asc nulls first
cont> ;
 EMPLOYEE_ID   SALARY_START   SALARY_END    SALARY_AMOUNT
 00164         14-Jan-1983    NULL            $51,712.00
 00164          5-Jul-1980     2-Mar-1981     $26,291.00
 00164          2-Mar-1981    21-Sep-1981     $26,291.00
 00164         21-Sep-1981    14-Jan-1983     $50,000.00
4 rows selected
SQL>
SQL> select employee_id, salary_start, salary_end, salary_amount
cont>  from salary_history sh
cont>  where employee_id = '00164'
cont>  order by salary_end asc nulls last
cont> ;
 EMPLOYEE_ID   SALARY_START   SALARY_END    SALARY_AMOUNT
 00164          5-Jul-1980     2-Mar-1981     $26,291.00
 00164          2-Mar-1981    21-Sep-1981     $26,291.00
 00164         21-Sep-1981    14-Jan-1983     $50,000.00
 00164         14-Jan-1983    NULL            $51,712.00
4 rows selected
SQL>
SQL> select employee_id, salary_start, salary_end, salary_amount
cont>  from salary_history sh
cont>  where employee_id = '00164'
cont>  order by salary_end desc nulls first
cont> ;
 EMPLOYEE_ID   SALARY_START   SALARY_END    SALARY_AMOUNT
 00164         14-Jan-1983    NULL            $51,712.00
 00164         21-Sep-1981    14-Jan-1983     $50,000.00
 00164          2-Mar-1981    21-Sep-1981     $26,291.00
 00164          5-Jul-1980     2-Mar-1981     $26,291.00
4 rows selected
SQL>
SQL> select employee_id, salary_start, salary_end, salary_amount
cont>  from salary_history sh
cont>  where employee_id = '00164'
cont>  order by salary_end desc nulls last
cont> ;
 EMPLOYEE_ID   SALARY_START   SALARY_END    SALARY_AMOUNT
 00164         21-Sep-1981    14-Jan-1983     $50,000.00
 00164          2-Mar-1981    21-Sep-1981     $26,291.00
 00164          5-Jul-1980     2-Mar-1981     $26,291.00
 00164         14-Jan-1983    NULL            $51,712.00
4 rows selected
SQL>
```

4.1.14 Enhancements to RMU Unload After_Image (LogMiner) Interface

This release of Oracle Rdb adds support for the following new options for the RMU Unload After_Image (aka LogMiner) command.

- The RMU Unload After_Image command can now output an XML script containing the updates to the selected table or tables.
- When the FORMAT qualifier specifies one of DELIMITED_TEXT, DUMP or XML, the TRIM option can also be used to trim leading and/or trailing spaces.
- The SYMBOLS qualifier now accepts the keyword LOCAL (default) or GLOBAL. The default behavior is to generate local scope DCL symbols. When Symbols=GLOBAL is used, these symbols have global scope.

4.1.14.1 XML Option to FORMAT Qualifier

When using FORMAT=XML, the following options can also be specified:

- CHARACTER_ENCODING_XML

  When using RMU Unload After_Image Format=XML, the XML header record will, by default, use the character encoding "ISO-8859-1". For example, this will appear in the header of the XML file.

  ```
  <?xml version="1.0" encoding="ISO-8859-1"?>
  ```

  This encoding (ISO-8859-1) is Latin 1 and covers encoding of many European character sets. However, this encoding is not adequate if you use other character encoding for Asian languages, or languages not covered by this ISO Standard.

  This release of Oracle Rdb adds an option, CHARACTER_ENCODING_XML, that allows the command procedure to specify an alternate character encoding. For example, if you wish to have the XML header describe UTF8, then specify the qualifier /FORMAT=(XML,CHAR="utf-8").

  ```
  <?xml version="1.0" encoding="utf-8"?>
  ```

- DATA_XML_NULL

  This option accepts one of the following keywords which control the output of NULL column values: DROP, NIL_ATTRIBUTE, or EMPTY. If this option is not specified, the default is EMPTY as shown in the following example.

  This is a fragment of the XML data generated by RMU Unload After_Image with the qualifier /FORMAT=(XML,TRIM=TRAILING) defaulting to DATA_XML_NULL=EMPTY.

  ```
   <ROW>
    <RDB_LM_ACTION>M</RDB_LM_ACTION>
    <RDB_LM_RELATION_NAME>SAMPLE2</RDB_LM_RELATION_NAME>
    <RDB_LM_RECORD_TYPE>32</RDB_LM_RECORD_TYPE>
    <RDB_LM_DATA_LEN>65254</RDB_LM_DATA_LEN>
    <RDB_LM_NBV_LEN>3</RDB_LM_NBV_LEN>
    <RDB_LM_DBK>60:14417:0</RDB_LM_DBK>
    <RDB_LM_START_TAD>2020-05-11T10:20:50.88</RDB_LM_START_TAD>
    <RDB_LM_COMMIT_TAD>2020-05-11T10:20:50.89</RDB_LM_COMMIT_TAD>
    <RDB_LM_TSN>712</RDB_LM_TSN>
    <RDB_LM_RECORD_VERSION>1</RDB_LM_RECORD_VERSION>
    <IDENT>101</IDENT>
    <COMMENT/>
    <DETAILS/>
   </ROW>
  ```

If DATA_XML_NULL is specified as DROP, then that column's value is omitted from the XML record. If DATA_XML_NULL is specified as NIL_ATTRIBUTE, then the XML tag attribute for the column is output as xsi:nil="true".

- TRIM

  This option allows the data values for the columns to have trailing and leading spaces and horizontal tab characters removed from the columns.

4.1.14.2 TRIM Option

When the FORMAT selected is one of DELIMITED_TEXT, DUMP or XML, then RMU can be instructed to trim trailing and/or leading spaces and horizontal tab characters from the columns.

- The default when FORMAT is XML or DELIMITED_TEXT is no trimming. The default when FORMAT is DUMP is TRIM_TRAILING.
- The TRIM option is not compatible with FORMAT=BINARY and FORMAT=TEXT.
- One of the following keywords can be specified for TRIM: TRAILING, LEADING and BOTH. If TRIM is specified without qualification, then TRAILING is assumed.

The following example shows the use of format XML with the TRIM=BOTH option.

```
$       RMU/UNLOAD-
            /AFTER_IMAGE -
            /INCLUDE=ACTION:(COMMIT,DELETE,NOMODIFY)-
            /LOG-
            /TABLE=(name=SAMPLE2, output=SAMPLE3.DAT)-
            /FORMAT=(XML,TRIM=BOTH)-
            /ORDER_AIJ_FILES-
        USER1:[TESTER.LOGMINER]LOGMINER_DB -
        USER1:[TESTER.LOGMINER]AIJ_BU_*.BAIJ
$
```

Usage Notes

- When unloading rows with columns that have many trailing spaces, then using FORMAT=(XML,TRIM) or FORMAT=(DELIMITED_TEXT,TRIM) can, in some cases, reduce the size of the output file without loss of significant data.
- When dumping rows with long columns that have many trailing spaces, then using /FORMAT=(DUMP,TRIM) can significantly reduce the size of the output file. Therefore, RMU Unload After_Image implicitly enables TRIM=TRAILING.

4.1.14.3 SYMBOLS Qualifier

The SYMBOLS qualifier now accepts the keyword LOCAL (default) or GLOBAL. The default behavior is to generate local scope DCL symbols. When Symbols=GLOBAL is used, these symbols have global scope.

Arguments

Symbols
Symbols=LOCAL
Symbols=GLOBAL
NoSymbols

Specifies whether DCL symbols are to be created, indicating information about records extracted for each table.

The default is Symbols, which causes local symbols to be created. Use Symbols=GLOBAL to have RMU define global symbols. Use NoSymbols to prevent creation of any DCL symbols.

If a large number of tables are being unloaded, too many associated symbols may be created and the CLI symbol table space can become exhausted. The error message "LIB-F-INSCLIMEM, insufficient CLI memory" is returned in this case. Specify the Nosymbols qualifier to prevent creation of the symbols.

4.1.15 Named Partition Support for RESERVING Clause

This release of Oracle Rdb adds the ability to use named partitions in the RESERVING clause of the SET TRANSACTION or DECLARE TRANSACTION statements. In prior versions, only partition numbers were allowed.

The partition names might be system generated (as shown below for the EMPLOYEES_MAP from the MF_PERSONNEL database) or they can be defined as part of the CREATE STORAGE MAP statement.

The following example shows the partition numbers as well as the system generated partition names under the *Partition information for storage map* output.

```
SQL> show storage map employees_map
     EMPLOYEES_MAP
 For Table:        EMPLOYEES
 Placement Via Index:    EMPLOYEES_HASH
 Partitioning is:    UPDATABLE

 Partition information for storage map:
 Compression is:    ENABLED
  Partition: (1) SYS_P00079
   Storage Area: EMPIDS_LOW
  Partition: (2) SYS_P00080
   Storage Area: EMPIDS_MID
  Partition: (3) SYS_P00081
   Storage Area: EMPIDS_OVER

SQL>
```

Usage Notes

- The PARTITION clause accepts a list of partition names or a list of partition ordinal values. You may not mix numeric and named notations.
- Duplicate partition names in the RESERVING clause will cause an exception. Review the RESERVING clause and correct the partition names.

```
set transaction
    read write
    evaluating
        job_history_foreign1 at verb time
        ,salary_history_foreign1 at verb time
    reserving
        employees partition (SYS_P00080, SYS_P00080) for exclusive write
;
%RDB-E-BAD_TPB_CONTENT, invalid transaction parameters in the transaction
parameter block (TPB)
-RDMS-E-DUPPARTNAME, partition SYS_P00080 for table EMPLOYEES already used
```

– Unknown partition names in the RESERVING clause (which might occur due to a change in the storage map definition) will cause an exception. Use the SHOW STORAGE MAP statement to review the partition names.

```
create module mod_testing1a
 language sql
 procedure proc_xa ();
 begin not atomic
 set transaction
    read write
    evaluating salary_history_foreign1 at verb time
    reserving employees partition (SYS_P00080, "UNKNOWN", SYS_P00081)
             ,departments for protected write;
  commit;
 end;
end module;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-BAD_TPB_CONTENT, invalid transaction parameters in the transaction
parameter block (TPB)
-RDMS-F-PARTNEXTS, partition "UNKNOWN" does not exist in this map or index
"EMPLOYEES_MAP"
```

4.2 Obsolete Features

4.2.1 RMU Backup No Longer Supports HUFFMAN or LZSS Compression, Use ZLIB Instead

This release of Oracle Rdb removes the compression options HUFFMAN and LZSS from the RMU Backup and RMU Backup After_Journal commands.

These older compression algorithms are much slower than the default ZLIB compression. Orders of magnitude more CPU is required in some cases. If you receive a diagnostic as shown in the following example, then Oracle recommends accepting the default, changing the DCL command procedure, or RMU PLAN file to explicitly state ZLIB. Additionally, ZLIB compression allows the database administrator to determine levels of compression efficiency, from less time to more effective compression. Refer to Oracle Rdb RMU Reference Manual for more details.

The following example shows the new diagnostic reported by RMU.

```
$ rmu/backup-
    /disk_file-
    /list_plan=plan_l.plan-
    /execute-
    /compress=lzss-
    /parallel=Executor_Count=3 -
    sql$database -
    sav_l1.rbf,sav_l2.rbf,sav_l3.rbf
%RMU-E-NOTSUPFORVER, The function COMPRESSION.LZSS is not supported for Oracle
Rdb V7.4-100
-RMU-I-COMPUSEZLIB, use the default, or specify ZLIB compression
%RMU-F-FTL_BCK, Fatal error for BACKUP operation at 18-JUN-2020 15:45:23.27
$
```

Also note that neither HUFFMAN nor LZSS are accepted by the RMU Set After_ Journal Backups qualifier. That command has always required the preferred ZLIB algorithm for compression.

A
Optimizer Enhancements

A.1 Optimizer Enhancements

A.1.1 Changes and Improvements to the Rdb Optimizer and Query Compiler

These features fall generally under the title *query rewrite*, and allow the query compiler to present a simplified query for optimization and execution.

- CAST function elimination

  In most cases, CAST actions must be executed at runtime to convert from the source data type to that specified by the CAST function. However, in some cases, the Rdb query compiler can eliminate or replace the CAST function with a literal value during query compile. This saves CPU time as the action is performed just once rather than once per row processed.

  This replacement includes the following:

  - When CAST of DATE (ANSI), DATE (VMS) or TIMESTAMP data types is performed to a compatible type of DATE or TIMESTAMP, then in many cases the CAST operator is not required.
  - CAST of string literals to DATE (ANSI), DATE (VMS), TIME, TIMESTAMP and INTERVAL can be processed at compile time. For example, CAST('2013-1-1' AS DATE ANSI) is implicitly converted to a DATE literal DATE'2013-1-1'.
  - CAST of small integer values is now done by the compiler. For example, CAST(1 AS SMALLINT) can be performed at compile time.
  - CAST of fixed length (CHAR) literal strings to varying length strings (VARCHAR) is now processed by the compiler if the character set is the same and the target VARCHAR is long enough to hold the source string, as seen in the following example:

    ```
    CAST('TABLE' AS VARCHAR(31))
    ```

- Constant Folding

  Simple arithmetic expressions involving integer or floating point literals are evaluated by the query compiler. The overall effect is smaller executable code and some reduced CPU time for queries. FLOAT, REAL, and DOUBLE PRECISION values are combined to produce DOUBLE PRECISION results. Integer literals (with no fractional component) are combined to produce BIGINT results.

  The side effect is that some expressions may now return DOUBLE PRECISION or BIGINT results where in prior versions they produced smaller precision results. This should not affect applications which fetch values into different data types as Oracle Rdb will perform an implicit conversion.

This optimization includes the following:

  * Addition (+)
  * Subtraction (-)
  * Multiplication (*)
  * Division (/)

    Note that division is not performed at compile time if the divisor is a literal zero (0). Operations which are coded to explicitly divide by zero are probably expected to produce an error at runtime. Although using the SQL SIGNAL statement is now preferred, this technique has been used to terminate procedures when an incorrect input is encountered.

- Algebraic Rules

  Additive identity (zero) can be added to an expression without changing the value. The query compiler will eliminate the literal zero (0) from the expression.

  Multiply by zero will result in zero if the other operand is a not nullable expression. In this case, the expression will be replaced by zero.

  Multiplicative identity (one) can be multiplied by an expression without changing the value. The query compiler will eliminate the literal one (1) from the expression.

  The side effect is that some expressions may now return slightly different data types because the literal is no longer considered as part of the data type computation.

- Simple Predicate Elimination

  When predicates include comparison of simple expressions, then the query compiler will attempt to eliminate them from the query predicate. For example, WHERE ('A' = 'A') will be replaced by TRUE, WHERE (2 <> 2) will be replaced with FALSE, and so on.

- Not Nullable Aware

  The query compiler is now aware of which columns have a NOT NULL NOT DEFERRABLE constraint enabled. Additionally, this attribute is also implied from any PRIMARY KEY NOT DEFERRABLE constraints.

  Using this knowledge, the query compiler can reduce (prune) the query expression. This list defines the ways in which this can occur:

  * When IS NULL is applied to a not nullable column or expression, then this predicate is replaced with FALSE.
  * When IS NOT NULL is applied to a not nullable column or expression, then this predicate is replaced with TRUE.

  The side effect is that constraints for a table are now loaded for SELECT statements.

  This optimization can be disabled using the SET FLAGS statement, or the RDMS$SET_FLAGS logical name with the value NOREWRITE(IS_NULL). The default is REWRITE(IS_NULL).

- Replace comparisons with NULL

Queries that erroneously compare value expressions with NULL will now be replaced with a simplified UNKNOWN value. For example, a query that uses WHERE EMPLOYEE_ID = NULL will never find matching rows, because the results of the comparison (equals, not equals, greater than, less than, and so on) are always UNKNOWN.

This optimization can be disabled using the SET FLAGS statement, or the RDMS$SET_FLAGS logical name with the value NOREWRITE(UNKNOWN). The default is REWRITE(UNKNOWN).

- Predicate Pruning

  The AND, OR and NOT operators can be simplified if the logical expressions have been reduced to TRUE, FALSE or UNKNOWN expressions. Depending on the operation, the Rdb query compiler might be able to eliminate the Boolean operator and part of the expression.

  This optimization can be disabled using the SET FLAGS statement, or the RDMS$SET_FLAGS logical name with the value NOREWRITE(BOOLEANS). The default is REWRITE(BOOLEANS).

- CASE Expression Pruning

  The prior transformation will also be applied to the Boolean WHEN expressions of a conditional expression (CASE, DECODE, NULLIF, COALESCE, NVL, NVL2, SIGN, ABS, and so on).

  In some cases, the resulting conditional expression might resolve to an equivalent conditional expression with fewer branches (some WHEN ... THEN clauses being eliminated) or a simple expression with no conditional expression (all WHEN ... THEN clauses are eliminated).

- IN Operator Simplification

  The IN operator using a subquery looks similar to the EXISTS boolean expression but it differs in its handling of NULL values. If the query compiler knows that neither source field nor the value set contains NULL, then the EXISTS expression can replace the IN operator. The EXISTS expression generates a better query solution in almost all cases.

  This optimization can be disabled using the SET FLAGS statement, or the RDMS$SET_FLAGS logical name with the value NOREWRITE(IN_CLAUSE). The default is REWRITE(IN_CLAUSE).

In most cases, the results of these optimizations will be transparent to the application. However, database administrators that use SET FLAGS 'STRATEGY,DETAIL' will notice new notations in the displayed strategy.

The following examples show the types of likely results.

In this example, the logical expression (1 = 2) is replaced with FALSE, the logical expression (1 = 1) is replaced with TRUE and the predicate is reduced to just the IS NULL (aka MISSING) check.

```
SQL> select last_name
cont> from employees
cont> where ((1 = 1) and employee_id is null)
cont>        or
cont>       ((1 = 2) and employee_id = '00164');
Tables:
  0 = EMPLOYEES
Conjunct: MISSING (0.EMPLOYEE_ID)
Get     Retrieval sequentially of relation 0:EMPLOYEES
0 rows selected
```

If there existed a NOT NULL NOT DEFERRABLE constraint on the EMPLOYEE_ID column, the expression can be further reduced because the NOT NULL constraint means the IS NULL test is always FALSE.

```
SQL> alter table EMPLOYEES
cont>     alter column EMPLOYEE_ID
cont>         constraint NN_EMPLOYEE_ID
cont>             NOT NULL
cont>             NOT DEFERRABLE
cont> ;
SQL>
SQL> select last_name
cont> from employees
cont> where ((1 = 1) and employee_id is null)
cont>        or
cont>       ((1 = 2) and employee_id = '00164');
Tables:
  0 = EMPLOYEES
Conjunct: FALSE
Get     Retrieval sequentially of relation 0:EMPLOYEES
0 rows selected
SQL>
```

REWRITE Flag

The SET FLAGS statement and the RDMS$SET_FLAGS logical name can be used to enable or disable some of these rewrite actions. This flag primarily exists for Oracle to test the behavior of the query rewrite changes. It can be used by programmers to revert to pre-V7.3 behavior.

REWRITE enables each rewrite setting and NOREWRITE disables them. Additionally, keywords can be added to REWRITE and NOREWRITE to disable selective rewrite actions.

The following new keywords are added for this release of Oracle Rdb.

- BOOLEANS
- IN_CLAUSE
- IS_NULL
- UNKNOWN

A.1.2 Optimized NOT NULL Constraint Execution

This release of Oracle Rdb introduces a new mechanism to verify NOT NULL constraints which are executed immediately at statement end (that is NOT DEFERRABLE). This new mechanism is more efficient (uses less code and virtual memory) than mechanisms used in prior releases. The cost of the constraint check in these cases is a fixed cost with a very small incremental cost for each extra NOT NULL constraint. The NOT NULL requirement of PRIMARY KEY constraints are also checked in the same way.

In prior releases of Oracle Rdb, each NOT NULL constraint would require its own internal query and each would be evaluated serially against the row just inserted or updated.

The following example shows an INSERT into a simple table with STRATEGY flags enabled. As can be observed, the absence of the strategy display indicates that no optimized query was used to validate these constraints.

```
SQL> set flags 'strategy,detail(2),internal,request_name';
SQL>
SQL> insert into SAMPLE
cont>     default values;
%RDB-E-INTEG_FAIL, violation of constraint SAMPLE_PK caused operation to fail
-RDB-F-ON_DB, on database RDB$DEFAULT_CONNECTION
SQL>
SQL> insert into SAMPLE (iden)
cont>     values (0);
%RDB-E-INTEG_FAIL, violation of constraint SAMPLE_DAT_NOT_NULL caused operation
to fail
-RDB-F-ON_DB, on database RDB$DEFAULT_CONNECTION
SQL>
SQL> insert into SAMPLE
cont>     values (1, 'A');
~Sn: Constraint "SAMPLE_PK" evaluated (verb)
Tables:
  0 = SAMPLE
  1 = SAMPLE
Cross block of 2 entries  Q1
  Cross block entry 1
    Conjunct: 0.DBKEY = <var0>
    Firstn: 1
    Get     Retrieval by DBK of relation 0:SAMPLE
  Cross block entry 2
    Conjunct: <agg0> <> 1
    Aggregate-F2: 0:COUNT-SINGLE (<subselect>) Q2
    Index only retrieval of relation 1:SAMPLE
      Index name  SAMPLE_NDX [1:1]
        Keys: 0.IDEN = 1.IDEN
1 row inserted
SQL>
```

Note that any DEFERRABLE constraints will be executed as in prior versions.

A.1.3 New BITMAPPED SCAN Clauses Added to OPTIMIZE Clause

This release of Oracle Rdb allows the programmer to specify the clause OPTIMIZE FOR BITMAPPED SCAN as part of a query. This clause requests that the query optimizer attempt to use BITMAPPED SCAN if there exists multiple supporting indices in the query. The Rdb query optimizer may ignore this request if only one index is used or if no SORTED RANKED indices would be used to solve the query.

The following example shows the effect of using this new clause.

```
SQL> set flags 'strategy,detail(2)';
SQL>
SQL> select count(*)
cont>  from car
cont>  where make = 'holden'
cont>    and cyear = 1979
cont>    and colour = 'blue'
cont>    and (ctype = 'sedan' or ctype = 'wagon')
cont> optimize for bitmapped scan
cont> ;
Tables:
  0 = CAR
Aggregate: 0:COUNT (*) Q2
Leaf#01 BgrOnly 0:CAR Card=6047  Bitmapped scan
  Bool: (0.MAKE = 'holden') AND (0.CYEAR = 1979)
        AND (0.COLOUR = 'blue')
        AND ((0.CTYPE = 'sedan') OR (0.CTYPE = 'wagon'))
  BgrNdx1 IYEAR [1:1] Fan=97
    Keys: 0.CYEAR = 1979
  BgrNdx2 ICOLOUR [1:1] Fan=79
    Keys: 0.COLOUR = 'blue'
  BgrNdx3 IMAKE [1:1] Fan=79
    Keys: 0.MAKE = 'holden'
  BgrNdx4 ITYPE [(1:1)2] Fan=79
    Keys: r0: 0.CTYPE = 'wagon'
          r1: 0.CTYPE = 'sedan'

                    1
1 row selected
SQL>
```

In previous releases, the programmer would need to define the logical name RDMS$ENABLE_BITMAPPED_SCAN as 1, RDMS$SET_FLAGS as "BITMAPPED_SCAN", or use the SET FLAGS 'BITMAPPED_SCAN' statement in the application.

A.1.4 Query Optimization Improvements for IN Clause

The EXISTS and IN predicates can often be used interchangeably in queries to check for the existence of values in another result set. If possible, the EXISTS query should be the first preference because its structure allows for the best query optimization. However, the semantics of these predicates are not identical when NULL values are present in one or both tables, especially when used with the NOT operator. Care should be taken to ensure correct query behavior in such cases.

With this release of Oracle Rdb, the optimizer will attempt to transform the IN predicate to an EXISTS predicate when the source columns are known to be not nullable. Such a transformation will return the same results and additionally present a better query for optimization.

The following example shows the strategy selected for NOT IN when the optimization is not (or cannot be) applied.

```
SQL> select s.badge_number
cont>  from STAFF s
cont>  where s.badge_number NOT IN (select kb.badge_number from KNOWN_BADGES kb)
cont> ;
Tables:
  0 = STAFF
  1 = KNOWN_BADGES
Cross block of 2 entries  Q1
  Cross block entry 1
    Index only retrieval of relation 0:STAFF
      Index name  STAFF_I [0:0]
  Cross block entry 2
    Conjunct: <agg0> = 0
    Aggregate-F1: 0:COUNT-ANY (<subselect>) Q2
    Conjunct: MISSING (0.BADGE_NUMBER) OR MISSING (1.BADGE_NUMBER) OR (
              0.BADGE_NUMBER = 1.BADGE_NUMBER)
    Index only retrieval of relation 1:KNOWN_BADGES
      Index name  KNOWN_BADGES_I [0:0]
 BADGE_NUMBER
            4
1 row selected
SQL>
```

When the target columns (for example BADGE_NUMBER) in each table have a NOT DEFERRABLE constraint of the type PRIMARY KEY or NOT NULL, then the following strategy is used. The resulting strategy will likely result in faster query execution.

```
SQL> select s.badge_number
cont>  from STAFF s
cont>  where s.badge_number NOT IN (select kb.badge_number from KNOWN_BADGES kb)
cont> ;
Tables:
  0 = STAFF
  1 = KNOWN_BADGES
Conjunct: <agg0> = 0
Match    (Agg Outer Join)  Q1
  Outer loop
  Match_Key:0.BADGE_NUMBER
    Index only retrieval of relation 0:STAFF
      Index name  STAFF_I [0:0]
  Inner loop      (zig-zag)
  Match_Key:1.BADGE_NUMBER
  Index_Key:BADGE_NUMBER
    Aggregate-F1: 0:COUNT-ANY (<subselect>) Q2
    Index only retrieval of relation 1:KNOWN_BADGES
      Index name  KNOWN_BADGES_I [0:0]
 BADGE_NUMBER
            4
1 row selected
SQL>
```

This transformation is enabled by default but can be disabled using SET FLAGS 'NOREWRITE(IN_CLAUSE)' and re-enabled using SET FLAGS 'REWRITE(IN_CLAUSE)'.

This new feature was actually introduced in Oracle Rdb Release 7.3.1 but was inadvertently left out of the release notes.

A.1.5 Query Optimization Improvements for DATE ANSI Queries

In prior releases of Oracle Rdb, a query such as the following would not use the index on the source column because the CAST function obscured the column reference from the optimizer.

The following example shows a query that is required to select all the transaction records that appeared on a specific date. That is, the query wants to ignore the time portion during the query.

```
SQL> select posting_timestamp
cont>  from TRANSACTION_LOG
cont>  where cast (posting_timestamp as DATE ANSI) = date ansi'2015-3-19'
cont> ;
Tables:
  0 = TRANSACTION_LOG
Index only retrieval of relation 0:TRANSACTION_LOG
  Index name  TRANS_NXD [1:1]
    Keys: (0.POSTING_TIMESTAMP >= DATE '2015-03-19') AND (0.POSTING_TIMESTAMP <
          (DATE '2015-03-19' + INTERVAL '1' DAY))
 POSTING_TIMESTAMP
 19-MAR-2015 00:25:21.73
1 row selected
SQL>
```

The Oracle Rdb optimizer now detects that an index column is within the CAST function and rewrites such queries to expose the index column. As can be seen, this query now performs an index range retrieval for all values in the specified date/time range.

A.1.6 New "Index Counts" Optimization for SORTED Indices

In prior releases of Oracle Rdb, a special optimization was applied to SORTED RANKED indices that reduced the I/O and CPU overhead for counting values within an index. In this release of Oracle Rdb, a similar optimization has been implemented for SORTED indices. The main benefit of this optimization is to greatly reduce the CPU overhead for processing SORTED indices with duplicate values.

The following example shows the new strategy applied for COUNT(*), COUNT(column), and COUNT(DISTINCT column). Here the column being referenced is the leading segment of a SORTED index.

```
SQL> select count(*) from employees;
Tables:
  0 = EMPLOYEES
Aggregate: 0:COUNT (*) Q2
Index only retrieval of relation 0:EMPLOYEES
  Index name  MI_NDX [0:0]       Index counts

                  100
1 row selected
SQL> select count(middle_initial)
cont> from employees where middle_initial = 'A';
Tables:
  0 = EMPLOYEES
Aggregate: 0:COUNT (0.MIDDLE_INITIAL) Q2
Index only retrieval of relation 0:EMPLOYEES
  Index name  MI_NDX [1:1]       Index counts
    Keys: 0.MIDDLE_INITIAL = 'A'
```

```
                    4
1 row selected
SQL> select count(distinct middle_initial)
cont> from employees where middle_initial = 'A';
Tables:
  0 = EMPLOYEES
Aggregate: 0:COUNT (DISTINCT 0.MIDDLE_INITIAL) Q2
Index only retrieval of relation 0:EMPLOYEES
  Index name  MI_NDX [1:1]       Index distinct counts
    Keys: 0.MIDDLE_INITIAL = 'A'

                    1
1 row selected
SQL>
```

This optimization is enabled by default and controlled by the flag COUNT_SCAN. Use the SET FLAGS 'NOCOUNT_SCAN' statement to disable this optimization, if necessary.

A.1.7 New Optimizations for the SQL IN Predicate

As described in *Section 1.1.7, New Implementation of the SQL IN Predicate* , SQL now uses a more compact representation of the SQL IN predicate. This presents new opportunities for Oracle Rdb 7.4.1.3 and later to improve incoming queries during compile and optimization.

Note

This new functionality can be disabled by defining the logical name SQL$DISABLE_IN_CLAUSE.

```
$ define /system SQL$DISABLE_IN_CLAUSE "Y"
```

Use RMU /SHOW LOGICAL_NAME SQL$DISABLE_IN_CLAUSE /DESCRIPTION to read more information.

Duplicate Entries in IN list are Eliminated

The new implementation allows the Rdb query compiler to eliminate duplicate literals, parameters, and column references while processing the IN predicate.

This first example shows the prior release's encoding of the query when there are many duplicate values.

```
SQL> set flags 'strategy,detail(2)';
SQL>
SQL> select *
cont>  from T
cont>  where a in (1,9,2,9,3,9,4,9,5,9,6,9,7,9,8,9)
cont> ;
Tables:
  0 = T
Conjunct: (0.A = 1) OR (0.A = 9) OR (0.A = 2) OR (0.A = 9) OR (0.A = 3) OR (0.A
           = 9) OR (0.A = 4) OR (0.A = 9) OR (0.A = 5) OR (0.A = 9) OR (0.A = 6)
           OR (0.A = 9) OR (0.A = 7) OR (0.A = 9) OR (0.A = 8) OR (0.A = 9)
Get     Retrieval sequentially of relation 0:T
   .
   .
   .
... rows selected
SQL>
```

When this query is run using 7.4.1.3 (or later) shows the reduce query complexity.

```
SQL> select *
cont>  from T
cont>  where a in (1,9,2,9,3,9,4,9,5,9,6,9,7,9,8,9)
cont> ;
Tables:
  0 = T
Conjunct: (0.A = 1) OR (0.A = 9) OR (0.A = 2) OR (0.A = 3) OR (0.A = 4) OR (0.A
           = 5) OR (0.A = 6) OR (0.A = 7) OR (0.A = 8)
Get     Retrieval sequentially of relation 0:T
   .
   .
   .
... rows selected
SQL>
```

When an index is present then the number of index lookups is minimized.

```
SQL> select *
cont>  from T
cont>  where a in (1,9,2,9,3,9,4,9,5,9,6,9,7,9,8,9)
cont> ;
Tables:
  0 = T
Conjunct: (0.A = 1) OR (0.A = 9) OR (0.A = 2) OR (0.A = 3) OR (0.A = 4) OR (0.A
           = 5) OR (0.A = 6) OR (0.A = 7) OR (0.A = 8)
Index only retrieval of relation 0:T
  Index name  T_NDX [(1:1)9]
    Keys: r0: 0.A = 8
          r1: 0.A = 7
          r2: 0.A = 6
          r3: 0.A = 5
          r4: 0.A = 4
          r5: 0.A = 3
          r6: 0.A = 2
          r7: 0.A = 9
          r8: 0.A = 1
   .
   .
   .
... rows selected
SQL>
```

Note

Oracle has observed similar queries from customers that include many duplicate values, most likely generated through front-end query tools. This change was made to address these observed overheads.

Leading value-expression only evaluated once

In prior versions any complex expression used as the left-hand side of the IN predicate would be propagated through the equivalent OR and equal (=) expression and lead to excessive evaluation. External functions, SQL functions, subqueries, and so on would add to the cost of the query execution.

For example, a simple subquery would end up creating a multi-way join expression, one join leg for each value in the IN predicate list.

```
SQL> select *
cont>  from T
cont>  where (select b from S limit to 1 row) not in (1,9,2,9,3)
cont> ;
Tables:
  0 = T
  1 = S
  2 = S
  3 = S
  4 = S
  5 = S
Cross block of 6 entries  Q1
  Cross block entry 1
    Conjunct: <agg0> <> 3
    Aggregate: 0:VIA (5.B) Q6
    Firstn: 1
    Get     Retrieval sequentially of relation 5:S
  Cross block entry 2
    Conjunct: <agg1> <> 9
    Aggregate: 1:VIA (4.B) Q5
    Firstn: 1
    Get     Retrieval sequentially of relation 4:S
  Cross block entry 3
    Conjunct: <agg2> <> 2
    Aggregate: 2:VIA (3.B) Q4
    Firstn: 1
    Get     Retrieval sequentially of relation 3:S
  Cross block entry 4
    Conjunct: <agg3> <> 9
    Aggregate: 3:VIA (2.B) Q3
    Firstn: 1
    Get     Retrieval sequentially of relation 2:S
  Cross block entry 5
    Conjunct: <agg4> <> 1
    Aggregate: 4:VIA (1.B) Q2
    Firstn: 1
    Get     Retrieval sequentially of relation 1:S
  Cross block entry 6
    Get     Retrieval sequentially of relation 0:T
```

```
   .
   .
   .
... rows selected
SQL>
```

The query compiler now generates a single evaluation of this expression thus reducing the query execution cost.

```
SQL> select *
cont>  from T
cont>  where (select b from S limit to 1 row) in (1,9,2,9,3)
cont> ;
Tables:
  0 = T
  1 = S
Cross block of 2 entries  Q1
  Cross block entry 1
    Conjunct: (<agg0> = 1) OR (<agg0> = 9) OR (<agg0> = 2) OR (<agg0> = 3)
    Aggregate: 0:VIA (1.B) Q2
    Firstn: 1
    Get     Retrieval sequentially of relation 1:S
  Cross block entry 2
    Get     Retrieval sequentially of relation 0:T

   .
   .
   .
... rows selected
SQL>
```

B
RDO, RDBPRE and RDB$INTERPRET Features

B.1 RDO, RDBPRE and RDB$INTERPRET Features

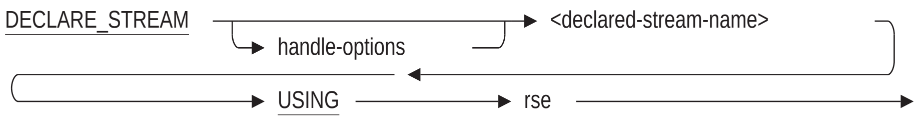
This section describes additions to the RDO and RDBPRE interfaces to Oracle Rdb. Please refer to the Rdb/VMS RDO Reference Manual which contains the latest definition of the RDO language.

B.1.1 New Request Options for RDO, RDBPRE and RDB$INTERPRET

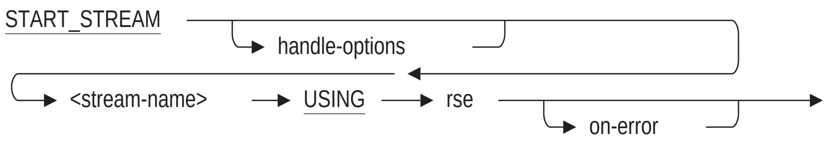
Two new keywords were added to the handle-options for the DECLARE_STREAM, the START_STREAM (undeclared format) and FOR loop statements.

The altered statements are shown below.

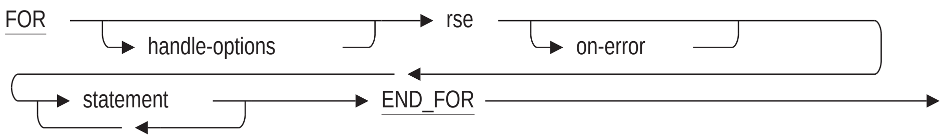
DECLARE_STREAM Format



START_STREAM Format

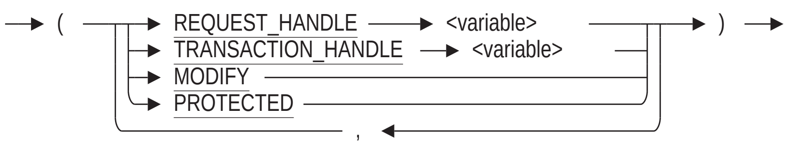


FOR Format



Each of these statements references the syntax for the HANDLE-OPTIONS which has been revised and is shown below.

handle-options =



The following options are available for HANDLE-OPTIONS:

- REQUEST_HANDLE specifies the request handle for this request. This option is only valid for RDBPRE and RDML applications. It cannot be used with RDB$INTERPRET, nor interactive RDO.
- TRANSACTION_HANDLE specifies the transaction handle under which this request executes. This option is only valid for RDBPRE and RDML applications. It cannot be used with RDB$INTERPRET, nor interactive RDO.
- MODIFY specifies that the application will modify all (or most) records fetched from the stream or for loop. This option can be used to improve application performance by avoiding lock promotion from SHARED READ for the FETCH to PROTECTED WRITE access for the nested MODIFY or ERASE statement. It can also reduce DEADLOCK occurrence because lock promotions are avoided.

  This option is valid for RDBPRE, RDB$INTERPRET, and interactive RDO. This option is not currently available for RDML.

  For example:

  ```
  RDO>   FOR (MODIFY) E IN EMPLOYEES WITH E.EMPLOYEE_ID = "00164"
  cont>    MODIFY E USING E.MIDDLE_INITIAL = "M"
  cont>    END_MODIFY
  cont>  END_FOR
  ```

  This FOR loop uses the MODIFY option to indicate that the nested MODIFY is an unconditional statement and so aggressive locking can be undertaken during the fetch of the record in the FOR loop.
- PROTECTED specifies that the application may modify records fetched by this stream by a separate and independent MODIFY statement. Therefore, this stream should be protected from interference (aka Halloween affect). The optimizer will select a snapshot of the rows and store them in a temporary relation for processing, rather than traversing indexes at the time of the FETCH statement. In some cases, this may result in poorer performance when the temporary relation is large and overflows from virtual memory to a temporary disk file, but the record stream will be protected from interference. The programmer is directed to the documentation for the Oracle Rdb logical names RDMS$BIND_WORK_VM and RDMS$BIND_WORK_FILE.

  This option is valid for RDBPRE, RDB$INTERPRET, and interactive RDO. This option is not currently available for RDML.

  The following example creates a record stream in a BASIC program using Callable RDO:

  ```
  RDMS_STATUS = RDB$INTERPRET ('INVOKE DATABASE PATHNAME "PERSONNEL"')

  RDMS_STATUS = RDB$INTERPRET ('START_STREAM (PROTECTED) EMP USING ' + &
                               'E IN EMPLOYEES')

  RDMS_STATUS = RDB$INTERPRET ('FETCH EMP')

  DML_STRING = 'GET ' +                                          &
                  '!VAL = E.EMPLOYEE_ID;' +                      &
                  '!VAL = E.LAST_NAME;' +                        &
                  '!VAL = E.FIRST_NAME' +                        &
                'END_GET'

  RDMS_STATUS = RDB$INTERPRET (DML_STRING, EMP_ID, LAST_NAME, FIRST_NAME)
  ```

  In this case, the FETCH needs to be protected against MODIFY statements which execute in other parts of the application.

The problem was corrected in Oracle Rdb Release 7.0.1.

B.1.2 New Language Features for RDO and Rdb Precompiler

The following new language enhancements have been made to RDO, the Rdb Precompiler (RDBPRE), and the RDO Interpreter (RDB$INTERPRET).

- LIKE operator

  ```
  --> <value_expr> LIKE <value_expr> ->
  ```

  The rse WITH clause can now specify a LIKE relational operator, which is similar in action to the MATCHING operator. The LIKE operator returns TRUE if the second expression pattern matches the first value expression. LIKE is case sensitive. LIKE uses these special characters:

  – % Matches any string
  – _ Matches any character
  – \ an escape character. Use \ \ to represent a single \ , \ % to represent a literal "%", and \ _ to represent a literal "_".

  This example is looking for any names starting with one character followed by an apostrophe.

  ```
  RDO> for e in employees
  cont>     with e.last_name like '_''%'
  cont>     print e.last_name
  cont> end_for
   LAST_NAME
   D'Amico
   O'Sullivan
  RDO>
  ```

- FIRST VIA ... FROM sub-query expression

  RDO includes a FIRST ... FROM sub-query expression. It returns the value from the matching row. However, if no rows are found, then the query will be aborted with a returned exception.

  The following example wishes to list each relation and its associated storage map (if it exists), and shows the reported RDB-E-FROM_NO_MATCH error.

  ```
  RDO> for r in rdb$relations
  cont>     with r.rdb$system_flag = 0
  cont>     sorted by r.rdb$relation_name
  cont>     print r.rdb$relation_name,
  cont>       first sm.rdb$map_name from sm in rdb$storage_maps with
  cont>             sm.rdb$relation_name = r.rdb$relation_name
  cont> end_for
   RDB$RELATION_NAME                 SM.RDB$MAP_NAME
   CANDIDATES                        CANDIDATES_MAP
  %RDB-E-FROM_NO_MATCH, no record matched the RSE in a "from" expression
  RDO>
  ```

  RDO now supports an alternative to the FIRST ... FROM sub-query expression which modifies the behavior when no matching rows were selected by the sub-query. Adding the VIA keyword requests that a MISSING value be returned in such cases, and the query is no longer aborted.

```
RDO> for r in rdb$relations
cont>     with r.rdb$system_flag = 0
cont>     sorted by r.rdb$relation_name
cont>     print r.rdb$relation_name,
cont>       first via sm.rdb$map_name from sm in rdb$storage_maps with
cont>             sm.rdb$relation_name = r.rdb$relation_name
cont> end_for
 RDB$RELATION_NAME                 SM.RDB$MAP_NAME
 CANDIDATES                        CANDIDATES_MAP
 COLLEGES                          COLLEGES_MAP
 CURRENT_INFO
 CURRENT_JOB
 CURRENT_SALARY
 DEGREES                           DEGREES_MAP
 DEPARTMENTS                       DEPARTMENTS_MAP
 EMPLOYEES                         EMPLOYEES_MAP
 EMPS
 JOBS                              JOBS_MAP
 JOB_HISTORY                       JOB_HISTORY_MAP
 RESUMES                           RESUMES_MAP
 SALARY_HISTORY                    SALARY_HISTORY_MAP
 WORK_STATUS                       WORK_STATUS_MAP
RDO>
```

- New special functions: RDO$CURRENT_USER, RDO$SESSION_USER and RDO$SYSTEM_USER

  These functions return the user identification of the current, session and system users. They can appear in any place that a field (aka column) can be used. These functions simplify view and trigger definitions created through RDO.

  Any view, computed by field, or trigger created by RDO but executed by a SQL session may return different values for each function. However, RDO sessions will typically return the same value from each function.

  This query uses the RDO$CURRENT_USER function to select the tables and views created by a user.

```
RDO> for r in rdb$relations
cont>     with r.rdb$relation_creator = rdo$current_user
cont>     print r.rdb$relation_name, r.rdb$created
cont> end_for
 RDB$RELATION_NAME                 RDB$CREATED
 EMKP                              23-JUN-2014 12:53:26.28
 PICK_HISTORY_REC                  31-JUL-2015 15:11:47.46
 TEST_TABLE                         8-AUG-2014 08:21:25.96
 CUSTOMER_REC                       8-AUG-2014 08:17:16.34
 TAB1                               8-AUG-2014 08:17:17.88
 TAB2                               8-AUG-2014 08:17:17.88
 JOB_HIST                          15-AUG-2014 13:49:07.39
 SAL_HIST                          15-AUG-2014 13:49:07.39
 EMP_NAMES                         14-OCT-2014 21:18:49.03
 CAND_NAMES                        14-OCT-2014 21:18:49.03
 ACTION_CODES                      14-OCT-2014 21:18:49.23
    .
    .
    .
```

B.1.3 RDO Interface Now Supports Synonym References

This release of Oracle Rdb adds minimal support for synonyms to RDO and RDBPRE. In prior versions, a synonym to a table (or view) was not recognized by the RDO interfaces. For instance, an application built against table names which were subsequently renamed using SQL would no longer compile because the synonyms established by the RENAME TABLE or ALTER TABLE ... RENAME TO statements were not recognized by RDBPRE or RDO.

This support allows queries that reference table or view synonyms to be processed by the RDBPRE precompiler and RDO interactive utility. In addition, most SHOW commands in RDO will recognize a table or view synonym.

Data definition (DDL) commands, such as DROP RELATION or DEFINE CONSTRAINT, do not accept a synonym name as input. For such operations, Oracle recommends using the Interactive SQL interface.

This problem has been corrected in Oracle Rdb Release 7.3.2.0. Synonyms for tables and views created using any of the following statements are now recognized by RDO.

- RENAME TABLE ...
- RENAME VIEW ...
- ALTER TABLE ... RENAME TO ...
- ALTER VIEW ... RENAME TO ...
- CREATE SYNONYMS ... FOR TABLE ...
- CREATE SYNONYMS ... FOR VIEW ...