Oracle® Rdb for OpenVMS

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Oracle® Rdb for OpenVMS

New Features Document

Release 7.1.x.x

June 2007

Oracle Rdb New Features, Release 7.1.x.x for OpenVMS

Contents

Preface

Purpose of This Manual

This manual contains the New Features Chapters for Oracle Rdb Release 7.1.5.1 and prior Rdb 7.1.x.x releases.

Document Structure

This manual consists of the following chapters:

Chapter 1	Describes enhancements introduced in Oracle Rdb Release 7.1.5.1.
Chapter 2	Describes enhancements introduced in Oracle Rdb Release 7.1.5.
Chapter 3	Describes enhancements introduced in Oracle Rdb Release 7.1.4.5.
Chapter 4	Describes enhancements introduced in Oracle Rdb Release 7.1.4.4.
Chapter 5	Describes enhancements introduced in Oracle Rdb Release 7.1.4.2.
Chapter 6	Describes enhancements introduced in Oracle Rdb Release 7.1.4.1.
Chapter 7	Describes enhancements introduced in Oracle Rdb Release 7.1.4.
Chapter 8	Describes enhancements introduced in Oracle Rdb Release 7.1.3.
Chapter 9	Describes enhancements introduced in Oracle Rdb Release 7.1.2.x.
Chapter 10	Describes enhancements introduced in Oracle Rdb Release 7.1.1.
Chapter 11	Describes enhancements introduced in Oracle Rdb Release 7.1.0.4.
Chapter 12	Sampled Selectivity Feature

Chapter 1
Enhancements Provided in Oracle Rdb Release 7.1.5.1

1.1 Enhancements Provided in Oracle Rdb Release 7.1.5.1

1.1.1 RMU Tape Support Added for SDLT600, LTO2, LTO3 Drives

Oracle Rdb RMU support has been added for the VMS tape density and compaction values for the Super DLT600, Ultrium460 and Ultrium960 tape drives. This will allow the following new density values to be specified with the /DENSITY qualifier for RMU commands that write to Super DLT600, Ultrium460 and Ultrium960 drives.

```
/DENSITY = (SDLT600,[NO]COMPACTION) − Super DLT600
/DENSITY = (LTO2,[NO]COMPACTION) − Ultrium460
/DENSITY = (LTO3,[NO]COMPACTION) − Ultrium960
```

The following shows examples of specifying density with or without compaction when backing up an Rdb database to one of these tape drives.

```
$ RMU/BACKUP/DENSITY=SDLT600/REWIND/LABEL=(LABEL1,LABEL2) −
 MF_PERSONNEL TAPE1:MFP.BCK, TAPE2:
$ RMU/BACKUP/DENSITY=(SDLT600,COMPACTION)/REWIND/LABEL=(LABEL1,LABEL2) −
 MF_PERSONNEL TAPE1:MFP.BCK, TAPE2:
$ RMU/BACKUP/DENSITY=LTO2/REWIND/LABEL=(LABEL1,LABEL2) −
 MF_PERSONNEL TAPE1:MFP.BCK, TAPE2:
$ RMU/BACKUP/DENSITY=(LTO2,COMPACTION)/REWIND/LABEL=(LABEL1,LABEL2) −
 MF_PERSONNEL TAPE1:MFP.BCK, TAPE2:
$ RMU/BACKUP/DENSITY=LTO3/REWIND/LABEL=(LABEL1,LABEL2) −
 MF_PERSONNEL TAPE1:MFP.BCK, TAPE2:
$ RMU/BACKUP/DENSITY=(LTO3,COMPACTION)/REWIND/LABEL=(LABEL1,LABEL2) −
 MF_PERSONNEL TAPE1:MFP.BCK, TAPE2:
```

1.1.2 New RMU VERIFY Messages %RMU−E−BADCLTSEQALLOC, %RMU−E−BADCLTSEQMAXID, %RMU−E−BADCLTSEQUSED

Three new diagnostic messages have been added to RMU/VERIFY for detecting Oracle Rdb database corruption when verifying Client Sequences. These messages will be output for inconsistencies detected between the client sequence definitions in the database root and the client sequence definitions in the RDB$SEQUENCES system table.

The %RMU−E−BADCLTSEQALLOC message is output if there is an inconsistency between the number of client sequences allocated in the database root and the number of client sequences defined in the system table RDB$SEQUENCES.

The %RMU−E−BADCLTSEQMAXID message is output if there is an inconsistency between the number of client sequences allocated in the database root and the maximum Sequence ID value defined in the system table RDB$SEQUENCES.

The %RMU−E−BADCLTSEQUSED message is output if there is an inconsistency between the number of client sequences in use in the database root and the number of client sequences defined in the system table RDB$SEQUENCES.

The following example shows all three of these new messages. The %RMU−E−NOSEQENT message is not a new message but an existing message already output by RMU/VERIFY.

```
$ RMU/VERIFY/ALL DISK:[DIRECTORY]MF_PERSONNEL
%RMU−E−BADCLTSEQALLOC, 32 client sequences allocated in the root is less
 than 55 client sequences defined in RDB$SEQUENCES.
%RMU−E−BADCLTSEQMAXID, 32 client sequences allocated in the root is less
 than the maximum client sequence id of 55 in RDB$SEQUENCES.
%RMU−E−NOSEQENT, sequence id 33 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 34 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 35 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 36 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 37 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 38 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 39 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 40 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 41 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 42 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 43 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 44 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 45 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 46 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 47 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 48 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 49 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 50 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 51 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 52 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 53 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 54 has no valid entry in the root file
%RMU−E−NOSEQENT, sequence id 55 has no valid entry in the root file
%RMU−E−BADCLTSEQUSED, 32 client sequences in use in the root does not
 equal 55 client sequences defined in RDB$SEQUENCES.
```

All three of these messages show database corruption that will require a database restore and recovery of the database to the last state that does not show this corruption.

1.1.3 Sample of Rdb External Routine Access to Oracle RDBMS

A set of files has been added to the SQL$SAMPLE directory which demonstrate the use of Rdb SQL external functions and procedures to access an Oracle RDBMS database. It includes PRO*C source code and build procedures along with an Rdb SQL script to define the external routines. The demonstration is composed of the following files:

- PRO_C_EXT_FUNC.COM
- PRO_C_EXT_FUNC.OPT
- PRO_C_EXT_FUNC.PC
- PRO_C_EXT_FUNC.SQL

The following interactive session shows how to build the shared executable and define the functions in a PERSONNEL database in an environment where ORAUSER.COM has been executed:

```
$ set default MY_DEMO_DIR
$ define PROCEXTFUNC MY_DEMO_DIR
$ copy sql$sample:PRO_C_EXT_FUNC.* *.*
$ @pro_c_ext_func.com

Pro*C/C++: Release 9.2.0.4.0 − Production on Fri May 11 19:34:32 2007

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

System default option values taken from: ora_proc20:pcscfg.cfg

 − Linking PRO_C_EXT_FUNC.EXE
$ SQL$
SQL> at 'f personnel';
SQL> @PRO_C_EXT_FUNC.SQL
SQL> commit;
SQL> exit;
$
```

The demonstration routines are designed to access the "EMP" table in the "SCOTT" schema in an Oracle RDBMS database. They allow data for this table to be retrieved, both with a singleton retrieval and using a cursor; to be updated; and to be inserted.

The demonstration creates an Rdb stored module named ORA_PRO_C_DEMO_FUNCS that contains the following external routines:

- roif_connect − a function
- roif_disconnect − a function
- roif_commit − a function
- roif_rollback − a function
- roif_get_errmsg − a procedure
- roif_get_employee − a procedure
- roif_open_emp_cursor − a function
- roif_fetch_emp_cursor − a procedure
- roif_close_emp_cursor − a function
- roif_update_employee − a procedure
- roif_insert_employee − a procedure

Chapter 2
Enhancements Provided in Oracle Rdb Release 7.1.5

2.1 Enhancements Provided in Oracle Rdb Release 7.1.5

2.1.1 Enhanced System Table Lookup in Multischema Databases

In prior releases of Oracle Rdb, applications that attached to a multischema database had to explicitly query the Rdb system tables using the catalog and schema name RDB$CATALOG.RDB$SCHEMA. Otherwise, a SET SCHEMA statement by the application might cause these system queries to fail. This was particularly a problem with interfaces such as SQL/Services and the Oracle ODBC Driver for Rdb.

With this release, Oracle Rdb will first try to locate the table in the default schema as established by the SET CATALOG, SET SCHEMA or ATTACH statements. If the lookup fails, Rdb will try RDB$CATALOG.RDB$SCHEMA. This lookup will apply to tables, sequences, functions and procedures for both system and user defined objects.

The following example shows the successful query with this new functionality.

```
SQL> attach 'filename db$:msdb';
SQL>
SQL> set schema 'west';
SQL>
SQL> select rdb$relation_name
cont> from rdb$relations
cont> where rdb$relation_name like 'JOB%';
RDB$RELATION_NAME
JOBS
JOB_HISTORY
2 rows selected
SQL>
```

The same query in an older version would fail.

```
SQL> attach 'filename db$:msdb';
SQL>
SQL> set schema 'west';
SQL>
SQL> select rdb$relation_name
cont> from rdb$relations
cont> where rdb$relation_name like 'JOB%';
%SQL−F−RELNOTDEF, Table RDB$RELATIONS is not defined in database or schema
SQL>
```

This problem has been corrected in Oracle Rdb Release 7.1.5.

2.1.2 Oracle Rdb Release 7.1.x.x New Features Document Name Changed

The file name of the Rdb Release 7.1.x.x New Features document has been changed from NEWFEATURES_71xx to RDB_NEWFEATURES_71xx in order to follow standard naming conventions.

Chapter 3
Enhancements Provided in Oracle Rdb Release 7.1.4.5

3.1 Enhancements Provided in Oracle Rdb Release 7.1.4.5

3.1.1 SHOW DOMAIN and SHOW TABLE Have Better Formatting of DEFAULT Strings

The output from the SHOW DOMAIN and SHOW TABLE command has changed with respect to the DEFAULT values that are strings. In prior versions, the default string was displayed without delimiters which made it hard to read, especially if the default value was all spaces. Additionally, strings from different character sets were not identified.

This release of SQL now displays these strings in quotes and prefixes it with the character set name, unless the character set is the session default.

The following example shows the revised output.

```
SQL> show domain STREET_NAME;
STREET_NAME                     CHAR(40)
 Oracle Rdb default: '>>'
SQL>
SQL> show table (column) PERSON;
Information for table PERSON

Columns for table PERSON:
Column Name                     Data Type        Domain
-----------                     ---------        ------
LAST_NAME                       CHAR(50)
 Oracle Rdb default: ' '
LATIN_NAME                      VARCHAR(30)
         ISOLATIN1 30 Characters,  30 Octets
 Oracle Rdb default: ISOLATIN1' '

SQL>
```

3.1.2 CALL Statement From Trigger Action Can Now Update Tables

Bug 2421356

In prior releases of Oracle Rdb, the CALL statement could only SELECT data from other tables. With this release of Rdb, the CALL statement may INSERT, DELETE and UPDATE tables as well as CALL other routines. The following restrictions apply to the actions of the routines activated by the CALL statement:

- The table which is the target for the trigger, known as the morphing table, may not be updated (meaning INSERT, DELETE or UPDATE) by any stored procedure or function called within the scope of the trigger activation. Morphing table updates must be done within a trigger definition so that anomalies can be detected and avoided. Attempts to update the morphing tables will result in a runtime error such as the following:

```
%RDB−E−READ_ONLY_REL, relation T was reserved for read access; updates not
allowed
−RDMS−E−RTN_ERROR, routine "SET_LENGTH" generated an error during execution
−RDMS−F−INVTRGACT_STMT, invalid trigger action statement − can not modify
target table
```

As far as the stored procedure is concerned, the morphing table is a read−only.

- If a stored routine action causes a different trigger to be activated and that then causes the same routine to be called, then an error similar to the following will be raised:

```
%RDB−F−ACTIVE_RTN, routine "CAST_VALUE" is already active
−RDMS−E−NORECURSION, no recursive routine calls permitted
```

Note

A stored routine may only be called from a trigger if it has been analyzed by Oracle Rdb. This step is automatically done by CREATE and ALTER TRIGGER ... ADD statements. If the routine was not recently created in the database (since Oracle Rdb Release 7.0.6), then use the ALTER MODULE ... COMPILE option to recompile any routines.

3.1.3 Enhancement to SQLCA

The following enhancements have been made to the SQLCA with this release:

- The SQLCA field SQLERRM[0] is now updated with the statement type by the PREPARE statement for all dialects. These numeric codes are listed in the table below.
  In previous releases, SQLERRM[0] was set only for ORACLE LEVEL1 and ORACLE LEVEL2 dialects.
- If the statement being prepared is a SELECT statement containing an INTO clause, then SQLCA field SQLWARN6 will contain the character "I". Such singleton SELECT statements can be executed without using a cursor.

Table 3−1 SQLCA SQLERRM [0] Values

Symbolic Name+	Value	SQL Statement
	0	Statement is unknown
SQL_K_OCTRDB_CONNECT	−1	Rdb Connect
SQL_K_OCTRDB_ATTACH	−2	Rdb Attach
SQL_K_OCTRDB_DISCONNECT	−3	Rdb Disconnect
SQL_K_OCTRDB_CREATE_MODULE	−4	Rdb Create Module
SQL_K_OCTRDB_ALTER_MODULE	−5	Rdb Alter Module
SQL_K_OCTRDB_DROP_MODULE	−6	Rdb Drop Module
SQL_K_OCTRDB_CREATE_DOMAIN	−7	Rdb Create Domain
SQL_K_OCTRDB_ALTER_DOMAIN	−8	Rdb Alter Domain
SQL_K_OCTRDB_DROP_DOMAIN	−9	Rdb Drop Domain
SQL_K_OCTRDB_CREATE_CATALOG	−10	Rdb Create Catalog

SQL_K_OCTRDB_ALTER_CATALOG	−11	Rdb Alter Catalog
SQL_K_OCTRDB_DROP_CATALOG	−12	Rdb Drop Catalog
SQL_K_OCTRDB_ALTER_SCHEMA	−13	Rdb Alter Schema
SQL_K_OCTRDB_DROP_SCHEMA	−14	Rdb Drop Schema
SQL_K_OCTRDB_SET_SESSION	−15	Rdb Set Session Authorization
SQL_K_OCTCTB	1	create table
SQL_K_OCTINS	2	insert
SQL_K_OCTSEL	3	select
SQL_K_OCTCCL	4	create cluster
SQL_K_OCTACL	5	alter cluster
SQL_K_OCTUPD	6	update
SQL_K_OCTDEL	7	delete
SQL_K_OCTDCL	8	drop cluster
SQL_K_OCTCIX	9	create index
SQL_K_OCTDIX	10	drop index
SQL_K_OCTAIX	11	alter index
SQL_K_OCTDTB	12	drop table
SQL_K_OCTCSQ	13	create sequence
SQL_K_OCTASQ	14	alter sequence
SQL_K_OCTATB	15	alter table
SQL_K_OCTDSQ	16	drop sequence
SQL_K_OCTGRA	17	grant
SQL_K_OCTREV	18	revoke
SQL_K_OCTCSY	19	create synonym
SQL_K_OCTDSY	20	drop synonym
SQL_K_OCTCVW	21	create view
SQL_K_OCTDVW	22	drop view
SQL_K_OCTVIX	23	validate index
SQL_K_OCTCPR	24	create procedure
SQL_K_OCTAPR	25	alter procedure
SQL_K_OCTLTB	26	lock table
SQL_K_OCTNOP	27	no operation
SQL_K_OCTRNM	28	rename
SQL_K_OCTCMT	29	comment
SQL_K_OCTAUD	30	audit
SQL_K_OCTNOA	31	noaudit
SQL_K_OCTCED	32	create database link
SQL_K_OCTDED	33	drop database link
SQL_K_OCTCDB	34	create database
SQL_K_OCTADB	35	alter database
SQL_K_OCTCRS	36	create rollback segment
SQL_K_OCTARS	37	alter rollback segment

SQL_K_OCTDRS	38	drop rollback segment
SQL_K_OCTCTS	39	create tablespace
SQL_K_OCTATS	40	alter tablespace
SQL_K_OCTDTS	41	drop tablespace
SQL_K_OCTASE	42	alter session
SQL_K_OCTAUR	43	alter user
SQL_K_OCTCWK	44	commit
SQL_K_OCTROL	45	rollback
SQL_K_OCTSPT	46	savepoint
SQL_K_OCTPLS	47	pl/sql execute
SQL_K_OCTSET	48	set transaction
SQL_K_OCTASY	49	alter system switch log
SQL_K_OCTXPL	50	explain
SQL_K_OCTCUS	51	create user
SQL_K_OCTCRO	52	create role
SQL_K_OCTDUS	53	drop user
SQL_K_OCTDRO	54	drop role
SQL_K_OCTSER	55	set role
SQL_K_OCTCSC	56	create schema
SQL_K_OCTCCF	57	create control file
SQL_K_OCTATR	58	alter tracing
SQL_K_OCTCTG	59	create trigger
SQL_K_OCTATG	60	alter trigger
SQL_K_OCTDTG	61	drop trigger
SQL_K_OCTANT	62	analyze table
SQL_K_OCTANI	63	analyze index
SQL_K_OCTANC	64	analyze cluster
SQL_K_OCTCPF	65	create profile
SQL_K_OCTDPF	66	drop profile
SQL_K_OCTAPF	67	alter profile
SQL_K_OCTDPR	68	drop procedure
SQL_K_OCTARC	70	alter resource cost
SQL_K_OCTCSL	71	create snapshot log
SQL_K_OCTASL	72	alter snapshot log
SQL_K_OCTDSL	73	drop snapshot log
SQL_K_OCTCSN	74	create snapshot
SQL_K_OCTASN	75	alter snapshot
SQL_K_OCTDSN	76	drop snapshot
SQL_K_OCTCTY	77	create type
SQL_K_OCTDTY	78	drop type
SQL_K_OCTARO	79	alter role
SQL_K_OCTATY	80	alter type

SQL_K_OCTCYB	81	create type body
SQL_K_OCTAYB	82	alter type body
SQL_K_OCTDYB	83	drop type body
SQL_K_OCTDLB	84	drop library
SQL_K_OCTTTB	85	truncate table
SQL_K_OCTTCL	86	truncate cluster
SQL_K_OCTCBM	87	create bitmapfile
SQL_K_OCTAVW	88	alter view
SQL_K_OCTDBM	89	drop bitmapfile
SQL_K_OCTSCO	90	set constraints
SQL_K_OCTCFN	91	create function
SQL_K_OCTAFN	92	alter function
SQL_K_OCTDFN	93	drop function
SQL_K_OCTCPK	94	create package
SQL_K_OCTAPK	95	alter package
SQL_K_OCTDPK	96	drop package
SQL_K_OCTCPB	97	create package body
SQL_K_OCTAPB	98	alter package body
SQL_K_OCTDPB	99	drop package body
SQL_K_OCTCDR	157	create directory
SQL_K_OCTDDR	158	drop directory
SQL_K_OCTCLB	159	create library
SQL_K_OCTCJV	160	create java
SQL_K_OCTAJV	161	alter java
SQL_K_OCTDJV	162	drop java
SQL_K_OCTCOP	163	create operator
SQL_K_OCTCIT	164	create indextype
SQL_K_OCTDIT	165	drop indextype
SQL_K_OCTAIT	166	reserver for alter indextype
SQL_K_OCTDOP	167	drop operator
SQL_K_OCTAST	168	associate statistics
SQL_K_OCTDST	169	disassociate statistics
SQL_K_OCTCAL	170	call method
SQL_K_OCTCSM	171	create summary
SQL_K_OCTASM	172	alter summary
SQL_K_OCTDSM	173	drop summary
SQL_K_OCTCDM	174	create dimension
SQL_K_OCTADM	175	alter dimension
SQL_K_OCTDDM	176	drop dimension
SQL_K_OCTCCT	177	create context
SQL_K_OCTDCT	178	drop context
SQL_K_OCTASO	179	alter outline

SQL_K_OCTCSO	180	create outline
SQL_K_OCTDSO	181	drop outline
SQL_K_OCTAOP	183	alter operator
SQL_K_OCTCEP	184	create encryption profile
SQL_K_OCTAEP	185	alter encryption profile
SQL_K_OCTDEP	186	drop encryption profile
SQL_K_OCTCSP	187	create spfile from pfile
SQL_K_OCTCPS	188	create pfile from spfile
SQL_K_OCTUPS	189	merge
SQL_K_OCTCPW	190	change password
SQL_K_OCTUJI	191	update join index
SQL_K_OCTASYN	192	alter synonym
SQL_K_OCTADG	193	alter disk group
SQL_K_OCTCDG	194	create disk group
SQL_K_OCTDDG	195	drop disk group
SQL_K_OCTALB	196	alter library
SQL_K_OCTPRB	197	purge user recyclebin
SQL_K_OCTPDB	198	purge dba recyclebin
SQL_K_OCTPTS	199	purge tablespace
SQL_K_OCTPTB	200	purge table
SQL_K_OCTPIX	201	purge index
SQL_K_OCTUDP	202	undrop object
SQL_K_OCTDDB	203	drop database
SQL_K_OCTFBD	204	flashback database
SQL_K_OCTFBT	205	flashback table

+The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb, therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

Chapter 4
Enhancements Provided in Oracle Rdb Release 7.1.4.4

4.1 Enhancements Provided in Oracle Rdb Release 7.1.4.4

4.1.1 Enhancements to the ALTER TABLE ... ALTER COLUMN Clause

Bugs 2170476 and 4874525

The ALTER COLUMN clause has been enhanced with this release of Oracle Rdb and now allows columns to be altered to and from COMPUTED BY, AUTOMATIC and IDENTITY special columns.

- ALTER COLUMN may now change an AUTOMATIC column to a normal updateable base column even if there exists constraints and indices, as long as the data types are the same. In prior releases, the presence of a database wide collating sequence prevented this action.
- A non−computed base column can now be altered to be an AUTOMATIC column. The old data is retained and the column is made read−only.
- A non−computed base column can now be altered to be a COMPUTED BY column. The old data will not be accessible (a warning is issued for interactive SQL) and references to that column will evaluate the COMPUTED BY expression. If indices or constraints reference this column, then the ALTER TABLE statement will fail.
  Note that altering the column back to a base, or automatic, column will allow older versions of the row data to be visible (any rows inserted while the column was a COMPUTED BY column will return NULL).
- Prior versions of Rdb allowed ALTER COLUMN to define a NOT NULL constraint for a COMPUTED BY column. In existing databases, this is harmless but Rdb now enforces the restriction that a COMPUTED BY column may not have constraints defined.
- The IDENTITY syntax is now supported by ALTER TABLE ... ALTER COLUMN clause.
  If the table has no existing IDENTITY column, a new sequence for the table will be created. Care must be taken to ensure that the IDENTITY will not generate existing values for the column as this would cause INSERT to fail. Use the parameters on IDENTITY to specify an appropriate START WITH value, or modify the sequence using ALTER SEQUENCE.
  If the table has an existing IDENTITY column then an error is raised.
- In some prior versions, a computed column (AUTOMATIC, IDENTITY) could be converted to a non−computed column. However, the dependency rows were never erased from the RDB$INTERRELATIONS table. This is now handled correctly. This is not a serious problem but it may cause unexpected errors when future ALTER and DROP statements are used for those referenced objects.
  Please contact Oracle Support for assistance if this problem arises.
- If an IDENTITY column is converted to a base column, a COMPUTED BY column, or AUTOMATIC column, then the special sequence is automatically dropped.
- If a column has a DEFAULT (base column or AUTOMATIC UPDATE AS column) and it is converted to a COMPUTED BY, AUTOMATIC AS or an AUTOMATIC INSERT AS column, then the DEFAULT value is removed (as these types of columns are incompatible with DEFAULT).

4.1.2 New SHOW STATISTICS Command for Interactive SQL

This release of Oracle Rdb adds a SHOW STATISTICS command to Interactive SQL. This command displays some simple process statistics for the current process and is used primarily to compare resource usage and elapsed time for different queries.

The following example shows the output after performing a typical query.

```
SQL> select count (*)
cont>  from employees natural full outer join job_history;

        274
1 row selected
SQL> show statistics;

                  process statistics at  5-MAR-2006 05:57:48.28
         elapsed time =   0 00:00:00.16              CPU time =   0 00:00:00.05
     page fault count = 430             pages in working set = 22768
   buffered I/O count = 26                  direct I/O count = 83
      open file count = 12              file quota remaining = 7988
           locks held = 138                  locks remaining = 16776821
      CPU utilization = 31.2%            AST quota remaining = 995
SQL>
```

The statistics are reset after each execution of SHOW STATISTICS.

4.1.3 RMU Load and Unload Now Support Table and View Synonyms

Bug 4018104

This release of Oracle Rdb adds support for table and view synonyms for the RMU Load and RMU Unload commands. In prior releases of Rdb, the synonym name was not understood by RMU and resulted in an error.

```
$ SQL$
SQL> show tables
User tables in database with filename db$:personnel
     CANDIDATES
     COLLEGES
     CURRENT_INFO                     A view.
     CURRENT_JOB                      A view.
     CURRENT_SALARY                   A view.
     DEGREES
     DEPARTMENTS
     EMPLOYEES
     JOBS
     JOB_HISTORY
     RESUMES
     SALARY_HISTORY
     WORK_STATUS
     EMPS                             A synonym for table EMPLOYEES
$ rmu/unload db$:personnel emps emps
%RMU-E-OUTFILDEL, Fatal error, output file deleted
-RMU-F-RELNOTFND, Relation (EMPS) not found
```

These tools now translate the synonym to the base object and process the data as though the base table had been named. This implies that the unload interchange files (.UNL), or record definition files (.RRD) that contain the table metadata will name the base table or view and not use the synonym name. Therefore, if the metadata is used against a different database, you may need to use the /MATCH_NAME qualifier to override this name during RMU Load.

4.1.4 Enhancements to Concurrent DDL Statements

Bug 4761143

In prior versions of Oracle Rdb, attempts to run several ALTER TABLE ... ADD CONSTRAINT commands in different sessions in parallel would either stall waiting for another transaction to finish or fail with a deadlock as shown in the following example.

```
SQL> alter table ORDER_LINES
cont>   add constraint ORDER_LINES_FK
cont>      foreign key (order_number) references orders (order_number)
cont>      not deferrable
cont> ;
%RDB-E-DEADLOCK, request failed due to resource deadlock
-RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-DEADLOCK, deadlock on client '........ORDE'
4544524F0000001F0000000400000055
```

This behavior occurs because of Rdb's locking of the target tables to ensure consistent data in all tables. For example, for the constraint in the example to validate, there must not be another transaction deleting rows from the ORDER_LINES table. Rdb ensures this by locking the metadata for each table referenced in a constraint definition.

To provide better concurrency, this release of Oracle Rdb now allows the following statements to be used when the target table is reserved in DATA DEFINITION mode.

- ALTER TABLE can now reference the table reserved for DATA DEFINITION MODE
  The following clauses are supported:
    - ADD CONSTRAINT
    - ALTER COLUMN ... CONSTRAINT
    - ENABLE CONSTRAINT, ENABLE PRIMARY KEY, and ENABLE UNIQUE (...)
    - DROP CONSTRAINT
    - ALTER COLUMN ... NULL
    - DISABLE CONSTRAINT, DISABLE PRIMARY KEY and DISABLE UNIQUE
    - DISABLE UNIQUE (...)

  The ADD and ENABLE CONSTRAINT are best suited to concurrent execution as they may require I/O to validate the constraint.
- ALTER INDEX can now be used to build all or part of the index on a table reserved for DATA DEFINITION mode.
  The following clauses are supported:
    - BUILD PARTITION and BUILD ALL PARTITIONS
    - REBUILD PARTITION and REBUILD ALL PARTITIONS
    - TRUNCATE PARTITION and TRUNCATE ALL PARTITIONS
    - COMMENT IS clause

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

- ALTER VIEW and CREATE VIEW may now reference a table reserved for DATA DEFINITION mode.
- COMMENT ON TABLE can now reference a table reserved for DATA DEFINITION mode.
- The statement DROP CONSTRAINT can now reference a constraint on a table reserved for DATA DEFINITION mode.

Note

In prior releases, only the CREATE INDEX statement was permitted within a transaction that reserved a table in DATA DEFINITION mode.

Most ALTER TABLE clauses are now supported for tables reserved for SHARED DATA DEFINITION. The exceptions are those clauses that change the structure of the table: ADD COLUMN, DROP COLUMN and ALTER COLUMN which changes the data type.

Chapter 5
Enhancements Provided in Oracle Rdb Release 7.1.4.2

5.1 Enhancements Provided in Oracle Rdb Release 7.1.4.2

5.1.1 /TRANSPORT Added to RMU/REPLICATE AFTER

Bug 4109344

The /TRANSPORT qualifier has been added to the RMU/REPLICATE AFTER START and CONFIGURE commands. This new qualifier allows the network transport to be specified. The valid values are "DECNET" and "TCPIP". The specified network transport is saved in the database.

In previous releases, to use TCP/IP as the network transport for Hot Standby, the system−wide logical "RDM$BIND_HOT_NETWORK_TRANSPORT" had to be defined.

See the following example of this new feature.

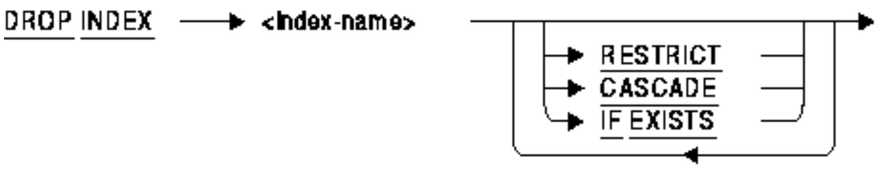
```
$ RMU/REPLICATE AFTER CONFIGURE /TRANSPORT=TCPIP
/STANDBY=REMNOD::DEV:[DIR]STANDBY_DB M_TESTDB
```

5.1.2 CASCADE Option Now Supported by DROP INDEX

In previous versions of Oracle Rdb, the option CASCADE was not supported. This meant that a DROP INDEX of an index used in the PLACEMENT VIA INDEX clause in a storage map would fail. The following example shows the generated error when the EMPLOYEES_HASH is dropped from the EMPLOYEES table in the MF_PERSONNEL database.

```
SQL> drop index EMPLOYEES_HASH;
%RDB−E−NO_META_UPDATE, metadata update failed
−RDMS−F−INDINMAP, index "EMPLOYEES_HASH" is used in storage map "EMPLOYEES_MAP"
SQL> drop index EMPLOYEES_HASH cascade;
SQL>
```

Syntax



Arguments

- CASCADE
  Specifies that you want SQL to modify any storage map that uses this index to be a NO PLACEMENT VIA INDEX storage map.

- IF EXISTS
  Prevents SQL command language from displaying error messages if the referenced index does not exist in the database.
- index−name
  Identifies the name of the index you wish to delete.
- RESTRICT
  Prevents the removal of an index if it is referenced by any other object within an Oracle Rdb database. RESTRICT is the default.

Usage Notes

- CASCADE will implicitly alter the storage map and change it to a NO PLACEMENT VIA INDEX storage map.
- Any query outline that references the index being dropped will be marked invalid for both RESTRICT and CASCADE options of the DROP INDEX command.
- In a multischema database, the DROP INDEX ... CASCADE statement will be used implicitly to support the DROP SCHEMA ... CASCADE statement. In previous versions of Oracle Rdb, this statement would fail if a storage map referenced an index that was to be dropped.

5.1.3 Oracle Rdb Release 7.1.x.x New Features Document Added

A new document has been created which contains all of the New Features Chapters from all previous Rdb 7.1 Release Notes. This document will be included in saveset A of the Rdb kit. It is called NEWFEATURES_71xx and will be available in postscript, text and PDF format. This will provide customers with one document to reference to find out about all new features that have been added to the Rdb 7.1 releases.

5.1.4 READER_THREAD_RATIO and RMU BACKUP

Prior to Oracle Rdb V7.1, RMU BACKUP created one reader thread to read each storage area in the storage area list that was assigned to a writer thread. This could cause a decrease in performance due to reader thread overhead caused by a larger number of reader threads competing for system resources, especially if a large number of storage areas was assigned to a writer thread.

Starting with Oracle Rdb Release 7.1.0, a reader thread pool has been used by default with the RMU BACKUP of a database to enhance performance by lessening the reader thread overhead caused by having a larger number of reader threads executing at the same time.

By default, 5 reader threads are created and assigned to each writer thread to read the storage areas assigned to that thread for backup. If fewer than 5 areas are assigned to a writer thread, one reader thread is created to read each storage area.

If the user does not want to use the default of 5 reader threads for each writer thread on a database backup because this default does not give the best backup performance for a particular database the

```
/READER_THREAD_RATIO = n
```

qualifier has been added where "n" specifies the number of reader threads to be created and assigned to each writer thread. The default is 5 or the actual number of storage areas assigned to the writer thread if it is less than 5. If "0" is specified, a thread pool is not used. The pre V7.1 method is used where the number of reader threads created and assigned to a writer thread equals the number of storage areas that writer thread is writing to the backup media. Therefore, the minimum value for using a thread pool is 1. There is no specific maximum number. If the specified maximum number of reader threads exceeds the number of storage areas assigned to a writer thread, it will be reduced to equal that number of storage areas. Therefore, no reader threads will be created without work to do.

The following example shows /READER_THREAD_RATIO used with a database backup to specify that 2 reader threads are to be used with each writer thread instead of the default of 5.

```
$ RMU/BACKUP/NOLOG/READER_THREAD_RATIO=2 MF_PERSONNEL MFP.RBF
$ RMU/RESTORE/NEW/NOCDD MFP.RBF
%RMU-I-AIJRSTAVL, 0 after-image journals available for use
%RMU-I-AIJISOFF, after-image journaling has been disabled
%RMU-W-USERECCOM, Use the RMU Recover command. The journals are not available.
```

The following example shows /READER_THREAD_RATIO=0 used with a database backup to specify that one reader thread should be created for each storage area assigned to a writer thread and not a maximum of 5 (in other words, use the pre V7.1 behavior).

```
$ RMU/BACKUP/NOLOG/READER_THREAD_RATIO=0 MF_PERSONNEL MFP.RBF
$ RMU/RESTORE/NEW/NOCDD MFP.RBF
%RMU-I-AIJRSTAVL, 0 after-image journals available for use
%RMU-I-AIJISOFF, after-image journaling has been disabled
%RMU-W-USERECCOM, Use the RMU Recover command. The journals are not available.
```

Chapter 6
Enhancements Provided in Oracle Rdb Release 7.1.4.1

6.1 Enhancements Provided in Oracle Rdb Release 7.1.4.1

6.1.1 Support Added for ANSI C Comments

For several years, the ANSI C standard has included the "//" style comment in addition to the traditional "/* */" style comment. Until now, SQL$PRE/CC has only supported the later style. Support for the "//" style comment has been added.

The following example shows comments in the old and in the newly supported style.

```
/*
 * Traditional C style comments
 */
i0 += 1;             /* add one to i0 */

//
// Comments using the // style
//
i0 += 1;             // add one to i0
```

6.1.2 New Rdb Character Set GB18030

Oracle Rdb has introduced a new Chinese character set, GB18030, which is defined by the GB18030−2000 standard as used by the People's Republic of China. GB18030 encodes characters in sequences of one, two, or four octets. The following are valid octet sequences:

```
§       Single−octet: %X'00'−%X'7F '

§       Two−octet: %X'81'−%X'FE' + %X'40−%X'7E' %X'80−%X'FE';

§       Four−octet: %X'81'−%X'FE' + %X'30'−%X'39' %X'81'−%X'FE' +%X'30'−%X'39'
```

The single−octet characters comply with the standard GB 11383 (iISO 4873:1986). GB18030 has 1.6 million valid octet sequences.

As GB18030 contains single−octet ASCII characters, it may be used as an Identifier Character Set.

6.1.3 New GET DIAGNOSTICS Keyword

The following new keyword has been added to GET DIAGNOSTICS:

- TRACE_ENABLED
  Returns an INTEGER value to indicate if the TRACE flag has been enabled using the statement SET FLAGS 'TRACE' or by either of the logical names RDMS$SET_FLAGS or RDMS$DEBUG_FLAGS. A zero (0) is returned if the flag is disabled, otherwise a one (1) is returned to indicate that tracing is enabled.

The following example shows usage of this keyword in a compound statement.

```
SQL> declare :x integer;
SQL> begin
cont> get diagnostics :x = TRACE_ENABLED;
cont> end;
SQL> print :x;
           X
           0
SQL> set flags 'trace';
SQL> begin
cont> get diagnostics :x = TRACE_ENABLED;
cont> end;
SQL> print :x;
           X
           1
```

6.1.4 Buffer Memory Now Exported/Imported

Bug 4213762

The Recovery Journal setting for Buffer Memory is now EXPORTed/IMPORTed.

An example of the syntax follows:

```
create data file rujmem;
export data file rujmem into rujmem;
drop data file rujmem;
import data from rujmem file rujmem recovery journal (buffer memory is local);
```

This feature has been added in Oracle Rdb Release 7.1.4.1.

6.1.5 New RESTART WITH Clause for ALTER SEQUENCE

This release of Oracle Rdb, 7.1.4.1, includes support for the ALTER SEQUENCE ... RESTART WITH clause. RESTART WITH allows the database administrator to reset the sequence to a specified value. The value must be within the range of MINVALUE and MAXVALUE. This command requires exclusive access to the sequence. Once the ALTER SEQUENCE statement is successfully committed, applications using the sequence will start with a value based on the restarted value.

Note

TRUNCATE TABLE for a table with an IDENTITY column implicitly executed an ALTER SEQUENCE ... RESTART WITH on the sequence. Specify the MINVALUE if it is an ascending sequence or MAXVALUE if it is a descending sequence.

The following example shows the new RESTART WITH clause.

```
SQL> show sequence NEW_EMPLOYEE_ID
     NEW_EMPLOYEE_ID
 Sequence Id: 1
 Initial Value: 472
...
```

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

6.1.6 ALTER VIEW Statement

Description

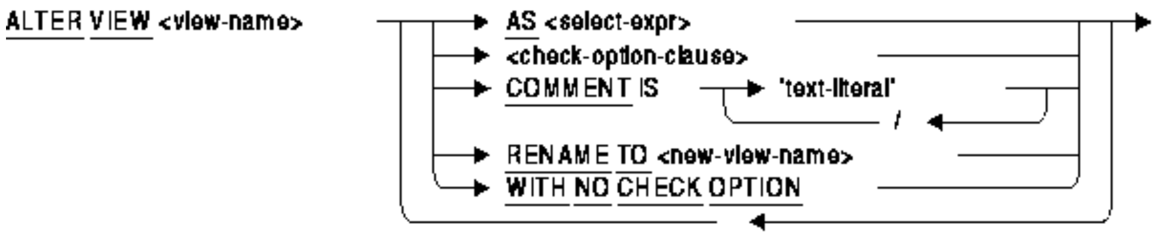
This statement allows the named view to be modified.

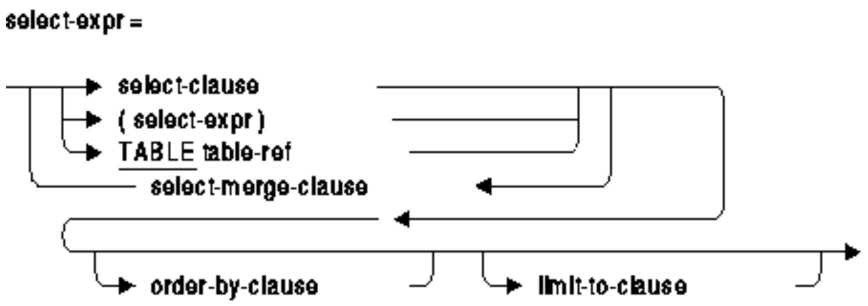
Environment

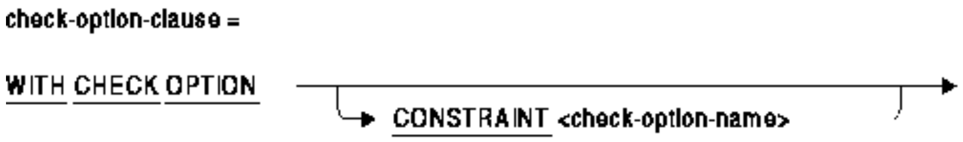
You can use the ALTER VIEW statement:

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

Format







Arguments

- AS
  Replaces the view select expression and the definitions of the columns. The number of expressions in the select list must match the original CREATE VIEW column list.
- CONSTRAINT check−option−name
  Specify a name for the WITH CHECK OPTION constraint. If you omit the name, SQL creates a name. However, Oracle Rdb recommends that you always name constraints. If you supply a name for the WITH CHECK OPTION constraint, the name must be unique in the schema.
  The name for the WITH CHECK OPTION constraint is used by the INTEG_FAIL error message when an INSERT or UPDATE statement violates the constraint.
- COMMENT IS
  Replaces the comment currently defined for the view (if any). The comment will be displayed by the SHOW VIEW statement in Interactive SQL.
- RENAME TO
  Renames the current view. The new view name must not exist as the name of an existing view, table, sequence or synonym.
- WITH CHECK OPTION
  A constraint that places restrictions on update operations made to a view. The check option clause ensures that any rows that are inserted or updated in a view conform to the definition of the view. Do not specify the WITH CHECK OPTION clause with views that are read−only. (The Usage Notes describe which views SQL considers read−only.)
- WITH NO CHECK OPTION
  Removes any check option constraint currently defined for the view.

Usage Notes

- You require ALTER privilege on the referenced view.
- The ALTER VIEW statement causes the RDB$LAST_ALTERED column of the RDB$RELATIONS table for the named view to be updated with the transaction's timestamp.

- Neither the column names nor their position may be modified using ALTER VIEW. Nor can columns be added or dropped for a view. These changes require both a DROP VIEW and CREATE VIEW statement to replace the existing definition.
- RENAME TO allows the name of the view to be changed. This clause requires that synonyms are enabled in the database. Use ALTER DATABASE ... SYNONYMS ARE ENABLED.
  The old name will be used to create a synonym for the new name of this view. This synonym can be dropped if the name is no longer used by database definitions or applications.
  This clause is equivalent to the RENAME VIEW statement.
- The COMMENT IS clause changes the existing comment on the view. This clause is equivalent to the COMMENT ON VIEW statement.
- Changes to the column expression may change the column to read−only and prevent referencing routines, triggers and applications from performing INSERT and UPDATE operations on those columns. Such changes will be reported at runtime.
  Similarly, if the view select table expression becomes read−only, referencing queries may fail.
  SQL considers as read−only views those with select expressions that:
    - Use the DISTINCT argument to eliminate duplicate rows from the result table
    - Name more than one table or view in the FROM clause
    - Use a derived table in the FROM clause
    - Include a statistical function in the select list
    - Contain a UNION, EXCEPT DISTINCT (MINUS), INTERSECT DISTINCT, GROUP BY, or HAVING clause
- If the AS clause changes the view to read−only, or includes a LIMIT TO ... ROWS clause on the main query then the check option constraint is implicitly removed.

Examples

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

Example 6−1 Changing the comment on a view

```
SQL> show view (comment) current_job
Information for table CURRENT_JOB

SQL> alter view CURRENT_JOB
cont>   comment is 'Select the most recent job for the employee';
SQL> show view (comment) current_job
Information for table CURRENT_JOB

Comment on table CURRENT_JOB:
Select the most recent job for the employee
SQL>
```

The following view uses a derived table and join to collect the count of employees in each department. The view is used in several reporting programs used by the department and company managers.

Example 6−2 Changing the columns results of a view definition

```
SQL> create view DEPARTMENTS_SUMMARY
cont> as
cont> select department_code, d.department_name,
cont>        d.manager_id, jh.employee_count
cont> from departments d inner join
cont>      (select department_code, count (*)
cont>         from job_history
cont>         where job_end is null
cont>         group by department_code)
cont>           as jh (department_code, employee_count)
cont>      using (department_code);
SQL>
SQL> show view DEPARTMENTS_SUMMARY;
Information for table DEPARTMENTS_SUMMARY

Columns for view DEPARTMENTS_SUMMARY:
Column Name                     Data Type        Domain
-----------                     ---------        ------
DEPARTMENT_CODE                 CHAR(4)
DEPARTMENT_NAME                 CHAR(30)
 Missing Value: None
MANAGER_ID                      CHAR(5)
 Missing Value:
EMPLOYEE_COUNT                  INTEGER
 Source:
select department_code, d.department_name,
       d.manager_id, jh.employee_count
from departments d inner join
     (select department_code, count (*)
        from job_history
        where job_end is null
        group by department_code) as jh (department_code, employee_count)
     using (department_code)

SQL>
```

The database administrator decides to create a column in the DEPARTMENTS table to hold the count of employees (rather than using a query to gather the total) and to maintain the value through triggers on EMPLOYEES and JOB_HISTORY (not shown here). Now the view can be simplified without resorting to a DROP VIEW and CREATE VIEW. Alter view will preserve the dependencies on the view from other views, triggers and routines and so minimize the work required to implement such a change.

```
SQL> alter table DEPARTMENTS
cont>     add column EMPLOYEE_COUNT integer;
SQL>
SQL> alter view DEPARTMENTS_SUMMARY
cont> as
cont> select department_code, d.department_name,
cont>        d.manager_id, d.employee_count
cont> from departments d;
SQL>
SQL> show view DEPARTMENTS_SUMMARY;
Information for table DEPARTMENTS_SUMMARY

Columns for view DEPARTMENTS_SUMMARY:
Column Name                     Data Type        Domain
-----------                     ---------        ------
DEPARTMENT_CODE                 CHAR(4)
 Missing Value: None
DEPARTMENT_NAME                 CHAR(30)
 Missing Value: None
MANAGER_ID                      CHAR(5)
 Missing Value:
EMPLOYEE_COUNT                  INTEGER
 Source:
select department_code, d.department_name,
       d.manager_id, d.employee_count
from departments d

SQL>
```

This example shows that a WITH CHECK OPTION constraint restricts the inserted data to the view's WHERE clause. Once the constraint is removed, the INSERT is no longer constrained.

Example 6−3 Changing the WITH CHECK OPTION constraint of a view definition

```
SQL> create view TOLIVER_EMPLOYEE
cont> as select * from EMPLOYEES where employee_id = '00164'
cont> with check option;
SQL> insert into TOLIVER_EMPLOYEE (employee_id) value ('00000');
%RDB−E−INTEG_FAIL, violation of constraint TOLIVER_EMPLOYEE_CHECKOPT1 caused
operation to fail
−RDB−F−ON_DB, on database DISK1:[DATABASES]MF_PERSONNEL.RDB;1
SQL>
SQL> alter view TOLIVER_EMPLOYEE with no check option;
SQL>
SQL> insert into TOLIVER_EMPLOYEE (employee_id) value ('00000');
1 row inserted
SQL>
```

Chapter 7
Enhancements Provided in Oracle Rdb Release 7.1.4

7.1 Enhancements Provided in Oracle Rdb Release 7.1.4

7.1.1 Support for OpenVMS Version 8.2

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

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

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

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

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

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

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

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

Plan Name = RDBBACKUP
Plan Type = BACKUP


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

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

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

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

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

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

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

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

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

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

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

```
$ RMU/UNLOAD/ROW_COUNT=5/COMMIT_EVERY=2/FLUSH=ON_COMMIT MF_PERSONNEL -
```

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

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

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

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

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

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

7.1.4 Altering COMPUTED BY and AUTOMATIC AS Columns

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

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

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

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

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

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

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

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

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

7.1.5 Enhancements to CREATE and ALTER STORAGE MAP

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

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

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

The mapping routine returns the following values:

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

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

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

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

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

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

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

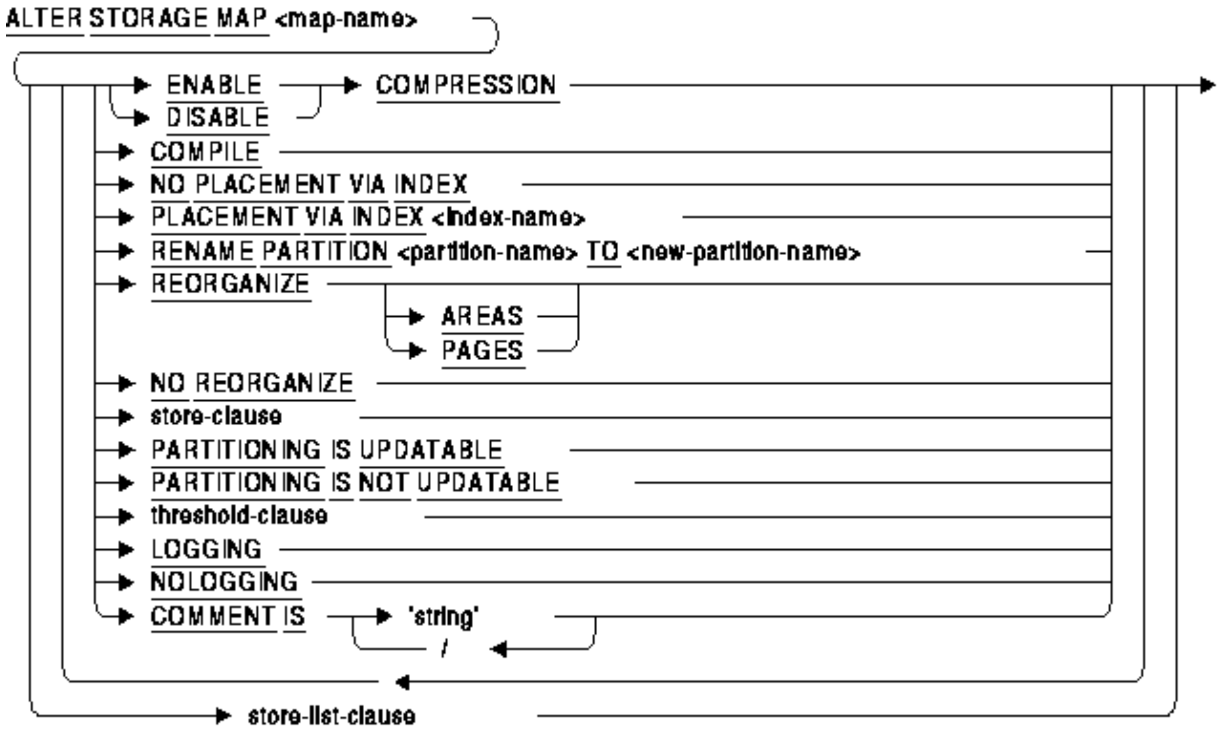
The ALTER STORAGE MAP command will remove any old mapping routine and redefine it when either the STORE clause is used or if the new COMPILE option is used.

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

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

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

Format


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

Note

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

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

7.1.6 RMU /UNLOAD /AFTER_JOURNAL /IGNORE OLD_VERSION Keyword

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

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

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

Chapter 8
Enhancements Provided in Oracle Rdb Release 7.1.3

8.1 Enhancements Provided in Oracle Rdb Release 7.1.3

8.1.1 Dynamic Optimizer FAST FIRST Shortcut Termination

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Enhancement 3633426

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

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

The following example shows the definition of the logical name.

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

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

```
MF_PERSONNEL_220456C4.LOG;1
```

8.1.3 Peephole Optimization for Hidden Key Retrieval

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

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

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

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

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

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

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

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

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

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

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

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

8.1.3.1 Feature Overview

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

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

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

```
 Smith
 Smith
2 rows selected
```

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

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

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

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

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

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

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

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

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

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

```
  Index name  EMP_LAST_NAME [0:0]
LAST_NAME
 Smith
 Smith
2 rows selected
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8.1.4 New PROTOTYPES Qualifier for SQL Module Language

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

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

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

8.1.5 RMU /UNLOAD /AFTER_JOURNAL Performance Enhancement

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

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

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

8.1.6 New GET DIAGNOSTICS Keywords

The following new keywords have been added to GET DIAGNOSTICS:

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

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

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

Dynamically executes a previously prepared statement.

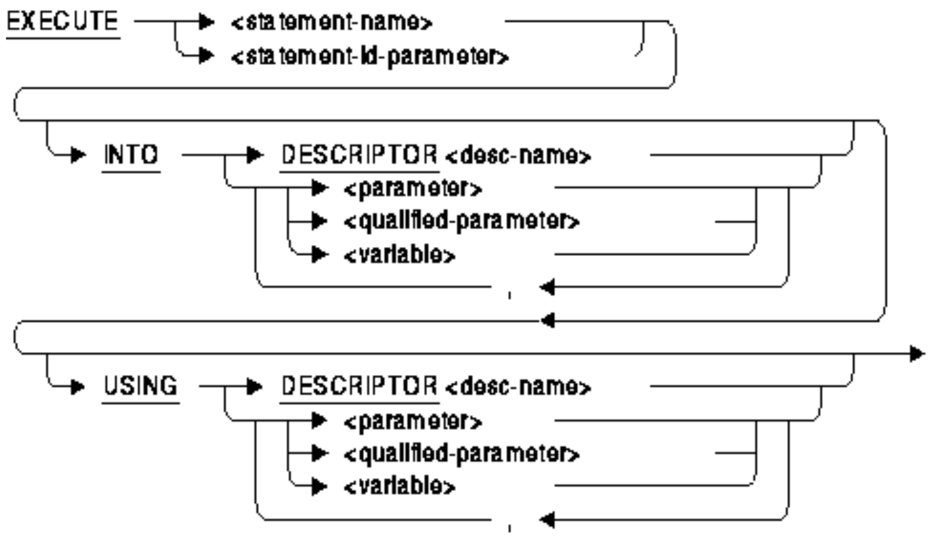
8.1.7 New EXECUTE Syntax

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

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

You can use the EXECUTE statement:

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



ARGUMENTS

INTO DESCRIPTOR descriptor−name

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

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

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

INTO parameter

INTO qualified−parameter

INTO variable

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

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

statement−name

statement−id−parameter

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

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

USING DESCRIPTOR descriptor−name

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

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

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

USING parameter

USING qualified−parameter

USING variable

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

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

Usage Notes

- You must use at least one USING or one INTO clause in an EXECUTE statement. If the statement has no parameters then use the EXECUTE IMMEDIATE statement instead.
- You may mix parameters with DESCRIPTOR structures within the EXECUTE statement. That is, you may use INTO DESCRIPTOR to hold the results of the dynamic statement, but use USING with a list of parameters to provide the input values.
- When you issue the EXECUTE statement for a previously prepared statement, you might want to obtain information beyond the success or failure code returned in the SQLCODE status parameter. For example, you might want to know how many rows were affected by the execution of an INSERT, DELETE, UPDATE, FETCH, or SELECT statement. SQL returns this information in the SQLERRD[2] field of the SQLCA.
  However, when you use an SQLCA parameter to prepare a statement, you must also use an SQLCA parameter when you execute that statement. For example, using SQL module language calls from C, your code might look like the following where the SQLCA parameter is passed to both procedures:

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

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

Examples

Example 1: Executing an INSERT statement with parameter markers

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

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

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


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

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

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

    mem_ptr output_save;
    mem_ptr input_save;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The SQL module language procedures called by the preceding fragment:

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

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

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

    PREPARE :DYN_STMT_ID FROM :STMT;

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

PROCEDURE DESCRIBE_SELECT
    SQLCA
    :DYN_STMT_ID    INTEGER
    SQLDA;

    DESCRIBE :DYN_STMT_ID OUTPUT INTO SQLDA;

PROCEDURE DESCRIBE_PARM
    SQLCA
    :DYN_STMT_ID INTEGER
    SQLDA;

    DESCRIBE :DYN_STMT_ID INPUT INTO SQLDA;


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

PROCEDURE EXECUTE_STMT
    SQLCA
    :DYN_STMT_ID INTEGER
    SQLDA;

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

8.1.8 Bitmap Scan Performance Enhancements

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

8.1.8.1 Bitmap Scan for OR Index Retrieval

Enhancement 3321352

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

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

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

```
SQL> create index sexi on employees (sex) type is sorted ranked;
SQL> create index addi on employees (address_data_1) type is sorted ranked;
SQL> commit;
SQL> set flags 'stratgey,detail'
SQL> select count(*) from employees
cont>     where address_data_1 starting with '1' or sex='M';
Tables:
  0 = EMPLOYEES
Aggregate: 0:COUNT (*)
OR index retrieval
  Conjunct: 0.ADDRESS_DATA_1 STARTING WITH '1'
  Get     Retrieval by index of relation 0:EMPLOYEES
    Index name  ADDI [1:1]
      Keys: 0.ADDRESS_DATA_1 STARTING WITH '1'
  Conjunct: NOT (0.ADDRESS_DATA_1 STARTING WITH '1')
  Get     Retrieval by index of relation 0:EMPLOYEES
    Index name  SEXI [1:1]
      Keys: 0.SEX = 'M'

          81
1 row selected
```

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

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

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

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

          81
1 row selected
```

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

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

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

8.1.8.2 Processing of Unique Keys and Non−ranked Indices

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

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

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

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

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

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

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

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

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

8.1.8.3 Enhanced Fast First Processing

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

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

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

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

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

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

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

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

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

2 rows selected
```

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

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

```
SQL> set flags 'strat,detail,exec,bitmap'
SQL> select * from t1 where f1=1 and f2=1;
```

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

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

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

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

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

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

8.1.9 Enhancements to SQL SHOW Commands

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

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

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

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

Information for table JOBS

Comment on table JOBS:
Possible jobs in the company

Information for table JOB_HISTORY

Comment on table JOB_HISTORY:
Employment history within the company

SQL>
```

Note

This support is not currently available for multischema databases.

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

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

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

Note

This support is not currently available for multischema databases.

8.1.10 Return EXCESS_TRAN with Distributed Transaction

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

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

To deal with this problem, Oracle Rdb has been enhanced so that if a new distributed transaction is started, it checks to see if any current one has begun the two phase commit (2PC) or rollback processing. If so, it enters a wait state until the previous transaction is complete (as previously). But, if the transaction is not being ended, it pauses for a while to see if the transaction end processing begins and, if it does not, it returns an %RDB−E−EXCESS_TRANS error. A new configuration file parameter, called

SQL_TRANS_START_WAIT, has been added to specify the number of seconds Rdb will wait for the end transaction processing before returning an %RDB−E−EXCESS_TRANS error. Rdb will pause in 500 millisecond intervals and recheck the existing transaction for up to the number of seconds specified by SQL_TRANS_START_WAIT, with a default of 3. SQL_TRANS_START_WAIT is defined in the client configuration file but may be specified for both the client and server sides of a remote connection.

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

```
SQL_TRANS_START_WAIT 4
```

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

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

exec sql declare alias filename personnel;

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

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

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

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

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

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

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

8.1.11 Dynamic SQL Enhancements

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

- The EXECUTE statement now supports INTO with a list of output host variables. In prior versions, only an INTO SQLDA was supported. See Section 8.1.7 for more details.
- DECLARE is now supported for creating local variables. Refer to the Oracle Rdb SQL Reference Manual, DECLARE Variable Statement. This statement is currently described as available in Interactive SQL only but is now available for dynamic SQL applications. These local variables will exist until a successful UNDECLARE or until the image runs down.
- UNDECLARE is now supported for deleting local variables. Refer to the Oracle Rdb SQL Reference Manual, UNDECLARE Variable Statement. This statement is currently described as available in Interactive SQL only but is now available for dynamic SQL applications.
- The PREPARE statement now sets values in the SQLCA to report the number of input and number output parameters for a statement. These values allow memory to be allocated for input and output SQLDA structures.
  Assuming that the SQLERRD array is zero based, SQL will set SQLERRD[2] to the count of output parameters and SQLERRD[3] to the count of input parameters. The values may possibly be zero and CALL parameters of INOUT type will appear in both the input and output count.
  The following simple program shows the effect of the new SQLCA support.

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

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

exec sql
    include SQLCA;

char * s1 = "begin insert into work_status values (?, ?, ?);\
             select count(*) into ? from work_status; end";

void main ()
{
int i;
SQLCA.SQLERRD[2] = SQLCA.SQLERRD[3] = 1;
exec sql
    prepare stmt from :s1;
if (SQLCA.SQLCODE != 0) sql_signal ();
printf( "SQLCA:\n  SQLCODE:    %9d\n", SQLCA.SQLCODE);
for (i = 0; i < 6; i++)
    printf( "  SQLERRD[%d]: %9d\n", i, SQLCA.SQLERRD[i]);
}
```

The results below show that there are 3 input arguments and 1 output argument.

```
SQLCA:
  SQLCODE:            0
  SQLERRD[0]:         0
  SQLERRD[1]:         0
  SQLERRD[2]:         1
  SQLERRD[3]:         3
  SQLERRD[4]:         0
  SQLERRD[5]:         0
```

Please note that the SQLCA was not set prior to Rdb Release 7.1.3 so Oracle recommends that the SQLERRD[2] and SQLERRD[3] values be set to a known value (such as −1) prior to the PREPARE call. Then if the values remain as −1 the application must estimate the counts itself.

Chapter 9
Enhancements Provided in Oracle Rdb Release 7.1.2.x

9.1 Enhancements Provided in Oracle Rdb Release 7.1.2.4

9.1.1 Query Governor Enhanced to Timeout Executing Requests

Bugs 809183 and 3240288

This release of Oracle Rdb introduces the ability to timeout long running database requests. A database request may include SELECT, UPDATE, DELETE and INSERT statements executed singly or as part of a compound statement, and includes any constraints or triggers associated with such actions.

Some database requests may take an excessive amount of time or system resources to execute. This feature provides the ability to have a request return an error if the request exceeds a defined limit for CPU or elapsed time.

Oracle Rdb provides the following interfaces to this timeout facility.

- SQL Interface
  New SQL syntax has been added to support this ability. An updated syntax diagram for SET QUERY is shown in Section 9.1.2, SET QUERY Statement. This new syntax is supported from both interactive and dynamic SQL interfaces. The following example establishes a 5 minute query timeout for the session.

  ```
  SET QUERY EXECUTION LIMIT ELAPSED TIME 5 MINUTES;
  ```

  For smaller intervals, the keyword SECONDS can be used, and is the default unit for this statement.

  ```
  SET QUERY EXECUTION LIMIT CPU TIME 30 SECONDS;
  ```
- Logical Name Interface
  Additionally, logical names may be used to define the timeout values. For example, to limit any database request that requires more than one second of CPU time or two seconds of elapsed time the following logical names can be defined.

  ```
  $ DEFINE RDMS$BIND_QG_EXEC_CPU_TIMEOUT 1
  $ DEFINE RDMS$BIND_QG_EXEC_ELAPSED_TIMEOUT 2
  ```

  When the timeout limit has been exceeded, Oracle Rdb will return an ***RDB−E−EXQUOTA*** error with a secondary error of ***RDMS−E−MAXTIMLIM***. For example:

  ```
  SQL> DELETE FROM EMPLOYEES;
  %RDB−E−EXQUOTA, Oracle Rdb runtime quota exceeded
  −RDMS−E−MAXTIMLIM, query governor maximum timeout has been reached
  ```
- RMU/SHOW STATISTICS Interface
  Requests may also be canceled using the ***RMU/SHOW STATISTICS*** utility. If a particular database user has a database request that is taking an excessive amount of time, it may be canceled by a

database administrator. The ***RMU/SHOW STATISTICS*** tools menu provides a mechanism to cancel a long running request. The tools menu is invoked by the "!" command. That menu contains the option "Terminate request". When that option is selected, ***RMU/SHOW STATISTICS*** prompts for the target user's process identifier (PID) and stream identifier. (The stream identifier is necessary if the user has more than one attach in that database.) The target process is then told to abort the currently running request. When the request is terminated, an ***RDB−E−EXQUOTA*** error is returned to the application; the secondary error code is ***RDMS−E−REQCANCELED***. For example:

```
SQL> DELETE FROM EMPLOYEES;
%RDB−E−EXQUOTA, Oracle Rdb runtime quota exceeded
−RDMS−E−REQCANCELED, request canceled
```

If an application receives a timeout error, it may continue to execute new database requests. The current transaction does not have to be terminated. New requests are still subject to the declared timeout limits.

Note that this timeout applies to the execution of a single database request. The timer is restarted for each new database request. A compound statement or stored procedure is treated as a single database request therefore, if a compound or stored procedure contains multiple statements or calls to nested stored routines, then all of the statements will be treated as a single database request.

Timeouts are not nested. If a compound statement includes an external function call and that external function issues a database request, the new database request issued by the external function becomes the current request being timed. When that external function returns, the compound statement that invoked the external function is no longer being timed. The exception is when the external function uses the BIND ON SERVER clause to force a separate server process to be used for the function invocation.

The timer is implemented using a low−overhead polling mechanism. That means that the timer is not completely accurate. For example, if a timeout is declared to be five seconds, a request may timeout in as little as five seconds but may not timeout for as long as ten seconds.

Timeouts are not applied to external routines which do not execute queries on the database. For instance, a compute bound function doing no I/O will not execute the polling required to terminate the request. Such external routines will not be terminated even if it exceeds the timeout values specified. However, once control returns to the calling request then the expired timer will be detected and the request terminated.

Timeouts only apply to basic query and data manipulation statements. Timeouts do not apply to database operations such as data definition statements (DDL). For example, a CREATE INDEX or an ALTER DATABASE operation will not timeout.

9.1.2 SET QUERY Statement

The SET QUERY statement is used to control query execution within a SQL session.

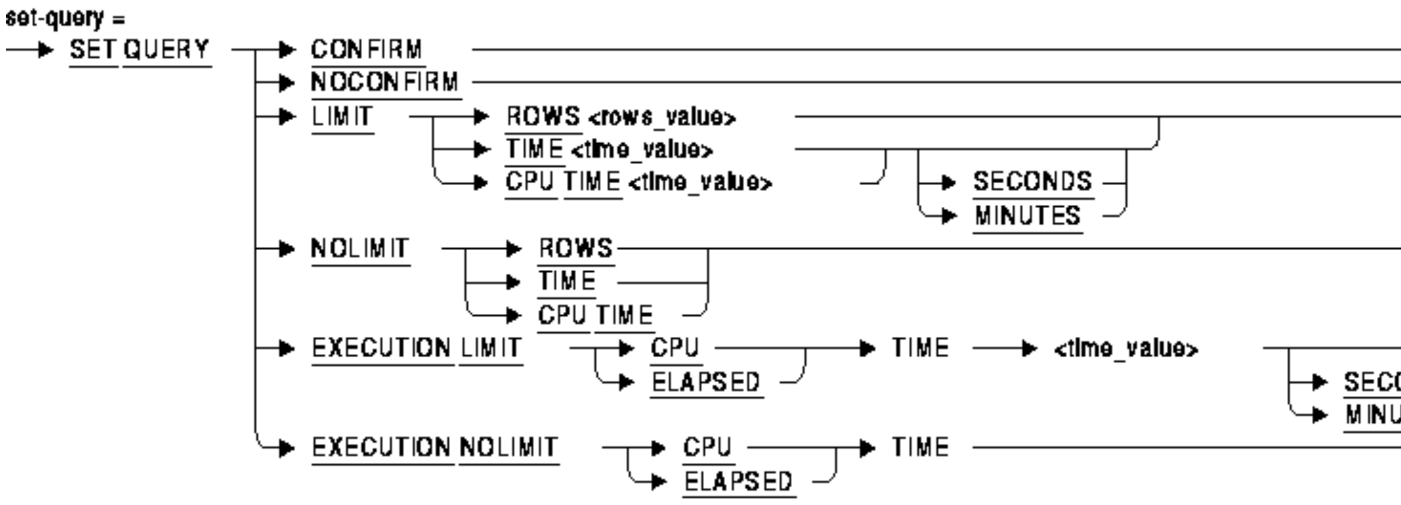
Environment

You can use the SET QUERY statement:

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

Note that some options for the SET QUERY command may only be used in interactive SQL.

Syntax:



Arguments:

- CONFIRM
  Lets you preview the cost of a query, in terms of I/O, before any rows are actually returned. For example:

  ```
  SQL> SELECT * FROM EMPLOYEES;
  Estimate of query cost: 52 I/O s, rows to deliver: 100
  Do you wish to cancel this query (No)? YES
  %SQL−F−QUERYCAN, Query cancelled at user's request
  ```

  Some queries can result in Oracle Rdb performing a large number of I/O operations, retrieving a large number of rows, or both. The SET QUERY CONFIRM statement causes SQL to display estimated query costs. If the cost appears excessive, you can cancel the query by answering Yes; to continue, answer No.
  The SET QUERY CONFIRM statement is only available for interactive SQL.
- NOCONFIRM
  Disables the query confirm dialog that was previously enabled using SET QUERY CONFIRM. The SET QUERY NOCONFIRM statement is only available for interactive SQL.
- LIMIT
  Sets limits to restrict the output generated by a query.
  The mechanism used to set these limits is called the query governor. The following gives you three ways to set limits using the query governor:
  - ♦ ROWS rows_value
    You can restrict output by limiting the number of rows a query can return. The optimizer counts each row returned by the query and stops execution when the row limit is reached.

The default is an unlimited number of row fetches. Dynamic SQL defaults are inherited from the compilation qualifier for the module.

  - ♦ TIME time_value [SECONDS | MINUTES]
    You can restrict the amount of time used to optimize a query for execution. If the query is not optimized and prepared for execution before the total elapsed time limit is reached, an error message is returned.
    The default is unlimited time for the query compilation. If you omit the SECONDS and MINUTES keyword then SECONDS is the default.

Note

Specifying a query time limit can cause application failure in certain circumstances. For instance, an application that runs successfully during off−peak hours may fail when run during peak hours due to the load on the database.

  - ♦ CPU TIME time_value [SECONDS | MINUTES]
    You can restrict the amount of CPU time used to optimize a query for execution. If the query is not optimized and prepared for execution before the CPU time limit is reached, an error message is returned.
    The default is unlimited CPU time for the query compilation. If you omit the SECONDS and MINUTES keyword then SECONDS is the default. Dynamic SQL options are inherited from the compilation qualifier for the module.

Use a positive integer for the number of rows and the number of seconds; negative integers are invalid and zero means no limits. If an established limit is exceeded, the query is canceled and an error message is displayed. When you set both a time limit and the row limit, whichever value is reached first stops the output.
Application developers can use this feature to prevent users from overloading the system. The database administrator can manage system performance and reduce unnecessary resource usage by setting option limits.

- NOLIMIT
  This option removes a limit imposed by the SET QUERY LIMIT command.
  Use one of the following options.
    - ♦ TIME
    - ♦ CPU TIME
    - ♦ ROWS

  NOLIMIT is equivalent to assigning a limit of zero to any of the options using SET QUERY LIMIT.
- EXECUTION LIMIT
  This option imposes elapsed and CPU time limits on executing queries. This command affects all subsequent queries executed within the Rdb server process. You must be attached to a database to execute this statement. This statement affects all attaches for the current process, not just the current connection.
    - ♦ ELAPSED TIME time_value [SECONDS | MINUTES]
    - ♦ CPU TIME time_value [SECONDS | MINUTES]

  You can restrict the amount of elapsed time or CPU time used to execute a query. If the query is not complete before the elapsed or CPU time limit is reached, an error message is returned.
  The default is unlimited time for the query execution. If you omit the SECONDS and MINUTES

keyword then SECONDS is the default. Dynamic SQL options are inherited from the compilation qualifier for the module.

Note

Specifying a query time limit can cause application failure in certain circumstances. For instance, an application that runs successfully during off−peak hours may fail when run during peak hours due to the load on the database.

Use a positive integer for the number of seconds or minutes; negative integers are invalid and zero means no limits. If an established limit is exceeded, the query is canceled and an error message is displayed. When you set a CPU time limit, elapsed time limit and a row limit (using SET QUERY LIMIT), whichever value is reached first stops the query.
Database administrators and application developers can use this feature to prevent users from overloading the system by executing long running, and probably unproductive queries. The database administrator can manage system performance and reduce unnecessary resource usage by setting option limits.

- EXECUTION NOLIMIT
  This option removes a limit imposed by the SET QUERY EXECUTION LIMIT command.
  Use one of the following options.
    - ♦ ELAPSED TIME
    - ♦ CPU TIME

  EXECUTION NOLIMIT is equivalent to assigning a limit of zero to any of the options using SET QUERY EXECUTION LIMIT.
- time_value
  This argument represents the number of seconds or minutes specified for the SET QUERY statement. It can be a numeric literal, a parameter name (for interactive SQL), or a parameter−marker (for dynamic SQL).
- rows_value
  This argument represents the number of rows specified for the SET QUERY statement. It can be a numeric literal, a parameter name (for interactive SQL), or a parameter−marker (for dynamic SQL).

Examples

This example shows the syntax for establishing a row limit within an interactive SQL session.

```
SQL> set query limit rows 10000;
SQL> show query limit;
QUERY LIMIT TIME is OFF
QUERY LIMIT ROWS limit is 10000 rows
QUERY LIMIT CPU TIME is OFF
SQL> set query nolimit rows ;
SQL> show query limit;
QUERY LIMIT TIME is OFF
QUERY LIMIT ROWS is OFF
QUERY LIMIT CPU TIME is OFF
```

This example uses SET QUERY to establish a 2 second elapsed time limit for a query, and shows the error message that is displayed.

```
SQL> set query execution limit elapsed time 2 seconds;
SQL> delete from EMPLOYEES;
%RDB−E−EXQUOTA, Oracle Rdb runtime quota exceeded
−RDMS−E−MAXTIMLIM, query governor maximum timeout has been reached
SQL> set query execution nolimit elapsed time;
```

9.2 Enhancements Provided in Oracle Rdb Release 7.1.2.3

9.2.1 Rdb Optional Site−Specific Startup Procedure

The Oracle Rdb startup procedure RMONSTART(xx).COM now supports an optional site−specific startup procedure to be executed after the Rdb Monitor (RDMMON) process has been started. If the file SYS$STARTUP:RDB$SYSTARTUP(xx).COM (where xx indicates the version number for multi−version Rdb kits) is found, it is executed as a DCL command procedure by the RMONSTART(xx).COM procedure.

SYS$STARTUP:RDB$SYSTARTUP(xx).COM is intended to contain site−specific tasks to be executed after the Rdb monitor procedure has completed. Such tasks might include opening databases or starting layered products that depend on the Rdb monitor process having been started.

If a site wishes to use this capability, the RDB$SYSTARTUP(xx).COM procedure must be created in SYS$STARTUP (either in the common SYS$COMMON:[SYS$STARTUP] directory or a node−specific SYS$SPECIFIC:[SYS$STARTUP] directory. The Rdb installation procedure does not provide or replace this file.

9.3 Enhancements Provided in Oracle Rdb Release 7.1.2

9.3.1 New NVL2 Expression

This release of Oracle Rdb adds a new conditional function, NVL2, which can simplify complex queries which require testing of values for NULL.

Syntax

NVL2 (value_expression, value_expression, value_expression)

Description

NVL2 lets you compute a value based on whether a specified expression is null or not null. If the first value expression is not null, then the second value expression is returned as the function result. Otherwise, the final value expression is returned. The data type of the NVL2 function is derived as a common data type of the second and third value expressions.

For example, when the JOB_END date in JOB_HISTORY is NULL then that indicates the current job for that employee. The following example uses NVL2 to annotate the output from a query on JOB_HISTORY displaying either "current job" or "prior job" based on the NULL attribute of the JOB_END column.

```
SQL> select employee_id, job_start, job_end,
cont> NVL2 (job_end, 'prior job', 'current job')
cont> from job_history
cont> where employee_id < '00180'
cont> order by employee_id, job_start;
 EMPLOYEE_ID   JOB_START     JOB_END
 00164          5-Jul-1980   20-Sep-1981   prior job
 00164         21-Sep-1981   NULL          current job
 00165          1-Jul-1975    4-Sep-1977   prior job
 00165          5-Sep-1977    7-Apr-1979   prior job
 00165          8-Apr-1979    7-Mar-1981   prior job
 00165          8-Mar-1981   NULL          current job
   .
   .
   .
```

The following example shows whether the income of some employees is made up of SALARY plus COMMISSION, or just SALARY, depending on whether the COMMISSION_PCT column of EMPLOYEES is null or not.

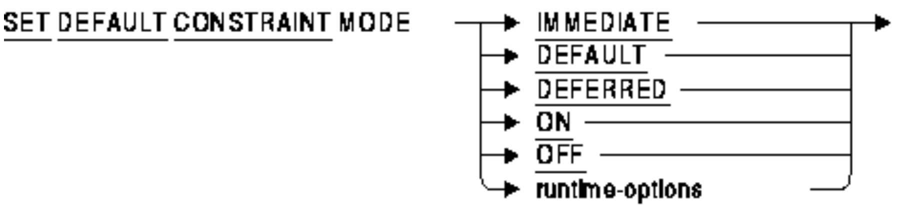
```
SQL> SELECT last_name, salary_amount,
cont>       NVL2 (commission_pct,
cont>             salary_amount + (salary_amount * commission_pct),
cont>             salary_amount) as Income edit using SALARY
cont> FROM employees e, salary_history sh
cont> WHERE last_name like 'B%'
cont>   and e.employee_id = sh.employee_id
cont>   and salary_end is null
cont> ORDER BY last_name;
```

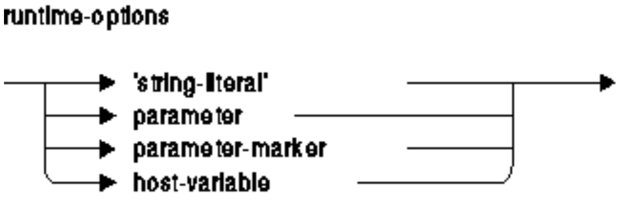
```
 E.LAST_NAME      SH.SALARY_AMOUNT        INCOME
 Babbin                $20,150.00    $20,956.00
 Bartlett              $14,817.00    $15,261.51
 Bartlett              $38,223.00    $38,987.46
 Belliveau             $54,649.00    $55,741.98
 Blount                $63,080.00    $64,341.60
 Boyd                  $30,275.00    $30,275.00
 Boyd                  $24,166.00    $24,166.00
 Brown                 $50,357.00    $50,357.00
 Burton                $23,053.00    $23,053.00
9 rows selected
SQL>
```

9.3.2 SET DEFAULT CONSTRAINT MODE Enhancements

This release of Oracle Rdb enhances the SET DEFAULT CONSTRAINT MODE statement. This statement is used to establish the session default constraint evaluation mode. In prior versions, this statement was only supported in Interactive SQL but it can now be used in Dyanmic SQL. In addition, the argument passed to this statement can now also be a parameter or host variable, allowing the value to be provided at runtime in both Dynamic and Interactive SQL.

Format





Arguments

- IMMEDIATE
  ON
  This requests that during the next transaction, all constraints defined as DEFERRABLE INITIALLY DEFERRED be evaluated as though defined as DEFERRABLE INITIALLY IMMEDIATE. ON is synonymous with IMMEDIATE.
- DEFAULT
  OFF
  This requests that during the next transaction, all constraints defined as DEFERRABLE INITIALLY

DEFERRED be evaluated as originally specified in the constraint definition. OFF is synonymous with DEFAULT.
- DEFERRED
  Currently this is synonymous with DEFAULT. However, in a future release of Oracle Rdb this keyword will change meaning.

Within a transaction, the constraint mode can be set temporarily using the SET ALL CONSTRAINTS statement. When a COMMIT or ROLLBACK is done, the mode will revert to that established by SET DEFAULT CONSTRAINT MODE.

If using runtime−options, the passed character value must be one of the keywords: ON, OFF, IMMEDIATE, DEFERRED, or DEFAULT. The following example shows how this can be done in Interactive SQL.

```
SQL> show constraint mode
    Statement constraint evaluation default is DEFERRED (off)
SQL> declare :c_mode char(10) = 'IMMEDIATE';
SQL> set default constraint mode :c_mode;
SQL> show constraint mode
    Statement constraint evaluation default is IMMEDIATE (on)
SQL>
```

9.3.3 New Dialect ORACLE LEVEL2 Added

This release of Oracle Rdb introduces a new dialect: ORACLE LEVEL2. This dialect is designed to be used in conjunction with SQL*Net for Rdb to provide better compatibility with the latest Oracle RDBMS releases.

ORACLE LEVEL2 includes all the semantics associated with ORACLE LEVEL1 with the following additions:

- Implicitly enabled SQL99 dialect semantics.
- Permits concatenation of non−character values. Each non−character value is implicitly CAST as a VARCHAR value before concatenation.
  Enhancement 2948230
- Date subtraction now results in a floating value with fractional portion. In ORACLE LEVEL1, the result is always a fixed point value with no fractional part.

Subsequent releases of Oracle Rdb may add to the list of features available in the ORACLE LEVEL2 dialect.

9.3.4 RMU/UNLOAD/AFTER_JOURNAL Output Flush

Bug 2832044

When using both the "OUTPUT" and "STATISTICS_INTERVAL" qualifiers with the RMU/UNLOAD/AFTER_JOURNAL command, the output stream used for the log, trace and statistics information is now flushed to disk (via the RMS $FLUSH service) at each statistics interval. This enhancement helps make sure that an output file of trace and log information is written to disk periodically.

9.3.5 RMU /SHOW STATISTICS Enhanced to Show Large Memory Setting

Bug 2903442

Previously, the RMU /SHOW STATISTICS "Buffer Information" screen did not display the setting of the global buffers "LARGE MEMORY IS ENABLED" setting.

This problem has been corrected in Oracle Rdb Release 7.1.2. The RMU /SHOW STATISTICS "Buffer Information" screen now indicates the database global buffers "LARGE MEMORY IS ENABLED" setting.

9.3.6 Statistics Collection Performance Improvement for AlphaServer GS Systems

NUMA (non−uniform memory access) is an attribute of a system in which access time to any given physical memory location is not the same for all CPUs. Given this architecture, consistently good location is important (but not necessarily 100 percent of the time) for highest performance. In the AlphaServer GS series, CPUs access memory in their own quad building block (QBB) faster than they access memory in another QBB. The OpenVMS operating system treats the hardware as a set of resource affinity domains (RADs). A RAD is a set of hardware components (CPUs, memory, and I/O) with common access characteristics. On AlphaServer GS80/160/320 systems, a RAD corresponds to a QBB.

Previously, a single copy of Oracle Rdb statistical information was maintained in a per−database memory structure (located in the database shared memory section). There was one copy of the statistical information for each database for all users on one OpenVMS system. Under heavy loads, the NUMA effect while maintaining statistics information could reduce the absolute performance of an application using Oracle Rdb due, in part, to increased memory access latency and CPU cache flushes.

The impact of this effect has been reduced. On AlphaServer GS series systems with more than one QBB configured with physical memory, the Oracle Rdb monitor process creates one global section per RAD that contains physical memory for the statistical information memory structure. These per−RAD global sections are created as "resident" and are requested to be allocated in physical memory by RAD. As each user attaches to the database, the user's OpenVMS defined "home" RAD is used to determine which global section to use for statistics collection for the user. The statistics global section is always mapped into the process' P0 virtual address space (ie, this global section is not controlled by SHARED MEMORY IS SYSTEM or LARGE MEMORY IS ENABLED).

Note

The global section creation requested in physical memory of a specific RAD is simply a "hint" to OpenVMS. Memory may be obtained from other RADs if no free memory is available at the time of allocation.

The RMU /SHOW STATISTICS utility maps all statistics global sections for a database. At each statistics collection interval, the statistical counters from each of the RAD−specific global sections are accumulated before display. Adding several copies of the statistics values together potentially increases the CPU consumption of the RMU /SHOW STATISTICS utility at each sample interval. However, the run−time performance gain by all database users should out−weigh the additional CPU cost of the RMU /SHOW

STATISTICS utility. Using a less−frequent update interval in the RMU /SHOW STATISTICS utility will result in less CPU consumption as well.

The virtual memory consumed by processes attached to databases, the Oracle Rdb monitor (RDMMON) and the RMU /SHOW STATISTICS utility will increase on those systems with more than one QBB containing physical memory. This is due to the mapping of multiple statistics shared memory global sections. However, because these sections are physically resident in memory, additional working set quota should not be required. The amount of additional virtual address space consumed is proportional to the number RADs configured in the system, the number of storage areas, the number of logical areas and the number of row caches configured in each database.

Note

OpenVMS support for RADs is available only on the AlphaServer GS series systems. For more information about using RADs, refer to the OpenVMS Alpha Partitioning and Galaxy Guide.

9.3.7 RMU RECOVER Accepts Wildcard After−image Journal File Specifications and ORDER_AIJ_FILES Qualifier

Bug 3032437

Starting with Oracle Rdb Release 7.1.2, the RMU /RECOVER command accepts wildcard after−image journal file specifications. File specifications containing the OpenVMS wildcard characters "%" and "*" are now parsed and processed.

By default, after−image journal files are processed in the order that they are presented to the RMU RECOVER command (either explicitly or as returned from OpenVMS when using wildcards). The new ORDER_AIJ_FILES qualifier specifies that the input after−image journal files are to be processed in ascending order by sequence number. This can be of benefit when you use wildcard (* or %) processing of a number of input files. The .AIJ files are each opened, the first block is read (to determine the sequence number), and the files are closed prior to the sequence number sorting operation.

9.3.8 RMU/SHOW STATISTICS Page Dump Content and Format Enhancements

Bug 2899761

Starting with Oracle Rdb Release 7.1.2, the RMU /SHOW STATISTICS utility displays the database page content, including the content of rows on the page in the "PageInfo" display, if the user has the OpenVMS BYPASS privilege enabled. The display of the page content is now also consistant with the output format of the RMU /DUMP command.

9.3.9 Enhancement to Prestarted Transaction Timeout

Bug 2439694

Oracle Rdb Release 7.1 introduced the ability to timeout "prestarted" transactions. That functionality has been enhanced to also force a process to obtain a new transaction sequence number (TSN) if the same TSN has been reused throughout the duration of the prestarted transaction timeout interval.

When a READ WRITE transaction does not make any modifications to the database, Oracle Rdb will reuse the TSN for the next transaction. If a user rarely or never makes any database modifications then the TSN for the user will become old. This can cause snapshot files to grow excessively. This enhancement provides the ability for processes that constantly reuse TSNs to periodically obtain a new TSN, thus preventing excessive snapshot growth.

This problem has been corrected in Oracle Rdb Release 7.1.2.

9.3.10 New SET CONTINUE CHARACTER Statement for Interactive SQL

Interactive SQL treats the minus sign (−) as the default continuation character. Unfortunately, this character is commonly used as the minus sign and if scripts are generated with minus at the end of the line then the results are unexpected.

In most cases, lines do not need to include a continuation character as most SQL statements are terminated with a ';' character and interactive SQL will continue to prompt until a valid statement is processed.

This release of SQL allows the database administrator to redefine the continuation character for interactive SQL. By selecting a little used character, the database administrator avoids problems with the minus sign and is still not required to use a continuation character in scripts. The SET CONTINUE CHARACTER statement could be included in the SQLINI file for the process.

The following example shows the effect of defining an alternate continuation character.

```
SQL> declare :val integer;
SQL> −− the trailing minus is assumed to be a continue character
SQL> begin
cont> set :val = :val −
cont>             (select count (*) from work_status);
             (select count (*) from work_status);
             ^
%SQL−F−LOOK_FOR, Syntax error, looking for ;, found ( instead
SQL> end;
end;
^
%SQL−F−LOOK_FOR_STMT, Syntax error, looking for a valid SQL statement, found END instead
SQL>
SQL> set continue character '\';
SQL> show continue character
Continue character is '\'.
SQL>
SQL> −− now the trailing minus is not used as a continue character
SQL> begin
cont> set :val = :val −
cont>             (select count (*) from work_status);
cont> end;
```

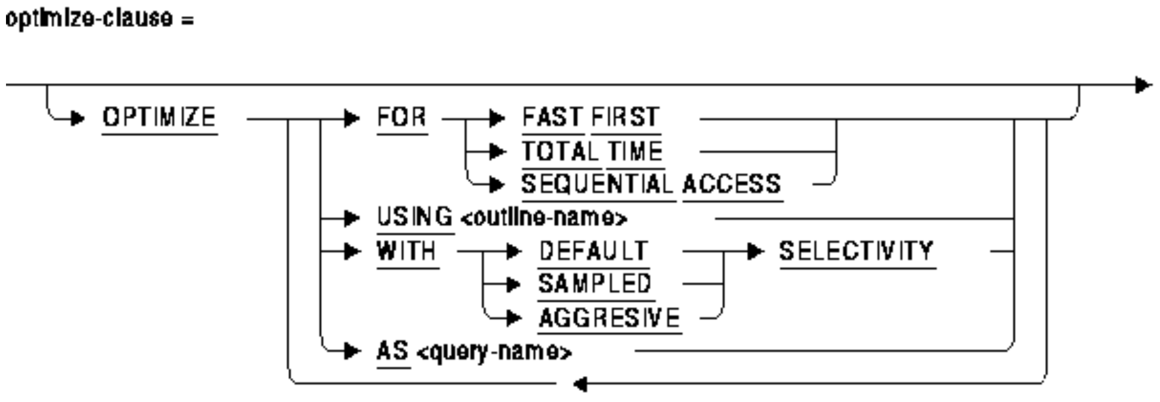
Usage Notes

- The continuation character must be a valid SQL language terminator. These characters are: '#', '(', ')', '*', '+', ',', '−', '.', '/', ':', ';', '?', '[', '\', ']', '{', |, and '}'.
- Currently only single octet values are supported by Interactive SQL.
- Use the SHOW CONTINUE CHARACTER to display the current continuation character.

9.3.11 OPTIMIZE Clause Enhancements

This release of Oracle Rdb introduces new controls for enabling optimizer selectivity using the OPTIMIZE clause on SELECT, INSERT ... SELECT, DELETE, UPDATE and compound statements (on the outermost BEGIN ... END).

Format



The enhanced syntax for OPTIMIZE now includes a WITH clause to select one of three optimization controls: DEFAULT (as used by previous versions of Rdb), AGGRESSIVE (assumes smaller numbers of rows will be selected), and SAMPLED (which uses literals in the query to perform preliminary estimation on indices).

The following example shows how to use this new clause.

```
SQL> select * from employees where employee_id < '00170'
cont>   optimize with sampled selectivity;
```

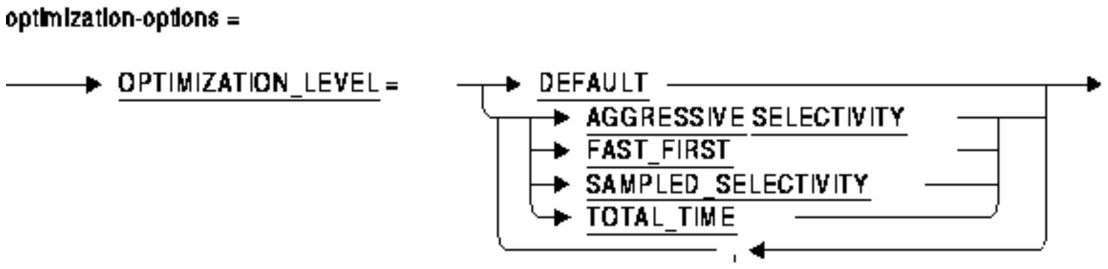
Please refer to Section 12.1.1 for more information.

9.3.12 New Options for the OPTIMIZATION_LEVEL Qualifier

This release of Oracle Rdb introduces new controls for enabling optimizer selectivity using the OPTIMIZATION_LEVEL qualifier on the SQL precompile or SQL Module Language compiler. This qualifier is used to establish default values for TOTAL TIME, FAST FIRST and the new SELECTIVITY clause. SELECT, INSERT ... SELECT, DELETE, UPDATE and compound statements (on the outermost BEGIN ... END) will inherit these settings during compile of the module.

See the Section 9.3.13 for similar functionality that can alter the defaults for Interactive SQL and Dynamic SQL.

Format



The enhanced syntax for the OPTIMIZATION_LEVEL qualifier now includes two new options: AGGRESSIVE_SELECTIVITY (assumes smaller numbers of rows will be selected), and SAMPLED_SELECTIVITY (which uses literals in the query to perform preliminary estimation on indices). You can choose one of these options and one of the options TOTAL_TIME or FAST_FIRST. Use a comma to separate the keywords and enclose the list in parentheses.

Only one of the options TOTAL_TIME or FAST_FIRST may be selected. Only one of the options AGGRESSIVE_SELECTIVITY or SAMPLED_SELECTIVITY may be selected. No other options may be include if DEFAULT is selected.

DEFAULT will set the module to the current Oracle Rdb defaults: FAST FIRST and DEFAULT SELECTIVITY.

The following example shows how to use this new functionality with the SQL Module Language compiler.

```
$ SQL$MOD/OPTIMIZATION_LEVEL=(TOTAL_TIME,SAMPLED_SELECTIVITY) APPCODE.SQLMOD
```

The following example shows how to use this new functionality with the SQL pre−compiler.

```
$ SQL$PRE/SQLOPTIONS=OPTIMIZATION_LEVEL=(TOTAL_TIME,SAMPLED_SELECTIVITY) APPCODE.SC
```

Note

Any query which explicily includes an OPTIMIZE WITH or OPTIMIZE FOR clause is not affected by the settings established using the OPTIMIZATION_LEVEL qualifier.

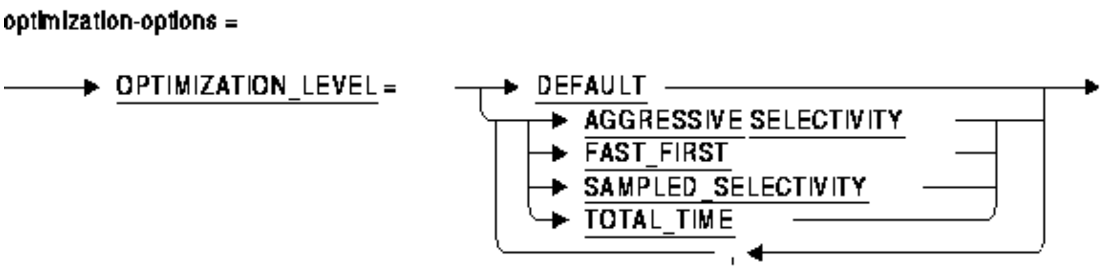
Please refer to Section 12.1.1 for more information.

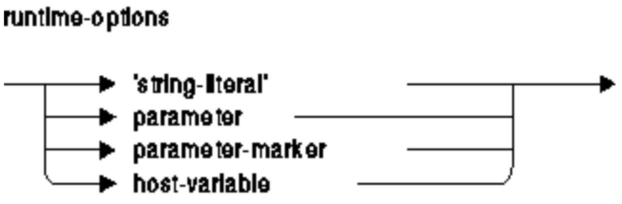
9.3.13 SET OPTIMIZATION LEVEL Enhancements

This release of Oracle Rdb introduces new controls for enabling optimizer selectivity using the SET OPTIMIZATION LEVEL statement. This statement is used to establish default values for TOTAL TIME, FAST FIRST and the new SELECTIVITY clause. Subsequent SELECT, INSERT ... SELECT, DELETE, UPDATE and compound statements (on the outermost BEGIN ... END) will inherit these settings in Interactive and Dynamic SQL.

See the Section 9.3.12 for similar functionality that can alter the defaults for SQL Module Language and the SQL precompiler.

Format





The enhanced syntax for SET OPTIMIZATION LEVEL now includes two new options: AGGRESSIVE SELECTIVITY (assumes smaller numbers of rows will be selected), and SAMPLED SELECTIVITY (which uses literals in the query to perform preliminary estimation on indices).

The runtime−options, either a literal value or parameter, can contain a formatted string containing the keyword DEFAULT, or one of TOTAL TIME or FAST FIRST, and one of AGGRESSIVE SELECTIVITY or SAMPLED SELECTIVITY. Use a comma to separate the clauses.

Only one of the options TOTAL TIME or FAST FIRST may be selected. Only one of the options AGGRESSIVE SELECTIVITY or SAMPLED SELECTIVITY may be selected. No other options may be included if DEFAULT is selected. A blank string will cause an error to be reported.

DEFAULT will set the session to the current Oracle Rdb defaults: FAST FIRST and DEFAULT SELECTIVITY.

The following example shows how to use this new functionality:

```
SQL> set optimization level 'total time, sampled selectivity';
SQL> select * from employees where employee_id > '00200';
```

Note

Any query which explicitly includes an OPTIMIZE WITH or OPTIMIZE FOR clause is not affected by the settings established using SET OPTIMIZATION LEVEL.

Please refer to Section 12.1.1 for more information.

9.3.14 RMU Unload Now Supports SELECTIVITY Option for OPTIMIZE Qualifier

The /OPTIMIZE qualifier for RMU Unload now supports a new SELECTIVITY qualifier so that the Rdb query optimizer can be influenced to use different selectivity values. Please refer to Section 12.1.1 for more details.

The SELECTIVITY option accepts the following keywords:

- AGGRESSIVE − assumes a smaller number of rows will be selected (compared to the default Rdb selectivity)
- SAMPLED − uses literals in the query to perform preliminary estimation on indices
- DEFAULT − uses default selectivity rules

The following example shows the new syntax.

```
$ RMU/UNLOAD/OPTIMIZE=(TOTAL_TIME,SELECTIVITY=SAMPLE) -
_$      SALES_DB CUSTOMER_TOP10 TOP10.UNL
```

This option is most useful when the RMU Unload command references a view definition with a complex predicate.

9.3.15 New Options Supported for LOGMINER SUPPORT Clause

The LOGMINER SUPPORT clause for CREATE DATABASE, IMPORT DATABASE, and ALTER DATABASE now allows the continuous mode for LogMiner to be enabled and disabled.

- LOGMINER SUPPORT IS ENABLED (CONTINUOUS)
  Enables continuous LogMiner.
- LOGMINER SUPPORT IS ENABLED (NOT CONTINUOUS)
  Disables continuous LogMiner, but leaves LogMiner enabled.
- LOGMINER SUPPORT IS DISABLED
  Disables LogMiner, including disabling continuous LogMiner.

Please refer to the Oracle Rdb RMU Reference Manual for more information about the Rdb LogMiner feature.

9.3.16 Changes to the IMPORT Command

In prior releases, the IMPORT command would display messages about the database. This is no longer true starting with Oracle Rdb Release 7.1.2. An example follows:

```
SQL> export data file mf_personnel into x;
SQL> drop data file mf_personnel;
SQL> import data from x file mf_personnel;
```

However, there is still the ability to generate these informational messages. A new clause, BANNER, has been added to the IMPORT command. Now, to enable informational messages to be displayed (hence the old behavior), simply specify BANNER on the IMPORT command line. Here is an example:

```
SQL> import data from x file mf_personnel BANNER;
Exported by Oracle Rdb X7.1−00 Import/Export utility
A component of Oracle Rdb SQL X7.1−00
Previous name was mf_personnel
It was logically exported on 29−MAY−2003 12:32
Multischema mode is DISABLED
Database NUMBER OF USERS is 50
Database NUMBER OF CLUSTER NODES is 16
Database NUMBER OF DBR BUFFERS is 20
Database SNAPSHOT is ENABLED
Database SNAPSHOT is IMMEDIATE
Database JOURNAL ALLOCATION is 512
Database JOURNAL EXTENSION is 512
Database BUFFER SIZE is 6 blocks
Database NUMBER OF BUFFERS is 20
Adjustable Lock Granularity is Enabled Count is 3
Database global buffering is DISABLED
Database number of global buffers is 250
Number of global buffers per user is 5
Database global buffer page transfer is via DISK
Journal fast commit is DISABLED
Journal fast commit checkpoint interval is 0 blocks
Journal fast commit checkpoint time is 0 seconds
Commit to journal optimization is Disabled
Journal fast commit TRANSACTION INTERVAL is 256
LOCK TIMEOUT is 0 seconds
Statistics Collection is ENABLED
Unused Storage Areas are: 0
System Index Compression is DISABLED
Journal was Disabled
Unused Journals are: 1
Journal Backup Server was:   Manual
Journal Log Server was:      Manual
Journal Overwrite was:       Disabled
Journal shutdown minutes was 60
Asynchronous Prefetch is ENABLED
Async prefetch depth buffers is 5
Asynchronous Batch Write is ENABLED
Async batch write clean buffers is 5
Async batch write max buffers is 4
Lock Partitioning is DISABLED
Incremental Backup Scan Optim uses SPAM pages
Unused Cache Slots are: 1
Workload Collection is DISABLED
Cardinality Collection is ENABLED
Metadata Changes are ENABLED
Row Cache is DISABLED
Detected Asynchronous Prefetch is ENABLED
Detected Asynchronous Prefetch Depth Buffers is 4
Detected Asynchronous Prefetch Threshold Buffers is 4
Open is Automatic, Wait period is 0 minutes
Shared Memory is PROCESS
Unused Sequences are: 32
The Transaction Mode(s) Enabled are:
        ALL
IMPORTing STORAGE AREA: RDB$SYSTEM
IMPORTing STORAGE AREA: DEPARTMENTS
IMPORTing STORAGE AREA: EMPIDS_LOW
IMPORTing STORAGE AREA: EMPIDS_MID
IMPORTing STORAGE AREA: EMPIDS_OVER
IMPORTing STORAGE AREA: EMP_INFO
IMPORTing STORAGE AREA: JOBS
IMPORTing STORAGE AREA: MF_PERS_SEGSTR
IMPORTing STORAGE AREA: SALARY_HISTORY
IMPORTing table CANDIDATES
IMPORTing table COLLEGES
IMPORTing table DEGREES
IMPORTing table DEPARTMENTS
IMPORTing table EMPLOYEES
IMPORTing table JOBS
IMPORTing table JOB_HISTORY
IMPORTing table RESUMES
IMPORTing table SALARY_HISTORY
IMPORTing table WORK_STATUS
IMPORTing view CURRENT_SALARY
IMPORTing view CURRENT_JOB
IMPORTing view CURRENT_INFO
```

It is also valid to specify NO BANNER on the IMPORT command line. Specifying NO BANNER is equivalent to specifying the IMPORT command with no clauses; that is informational messages about the database will not be displayed. In the following example, note that no informational messages are displayed:

```
SQL> import data from x file mf_personnel NO BANNER;
```

Warning and error messages will continued to be displayed as in prior releases.

9.3.17 ALTER MODULE Statement

Description

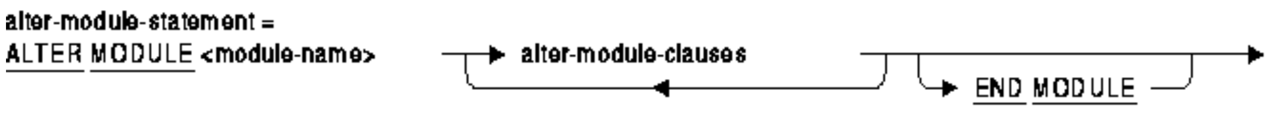
This statement allows the named module to be modified.

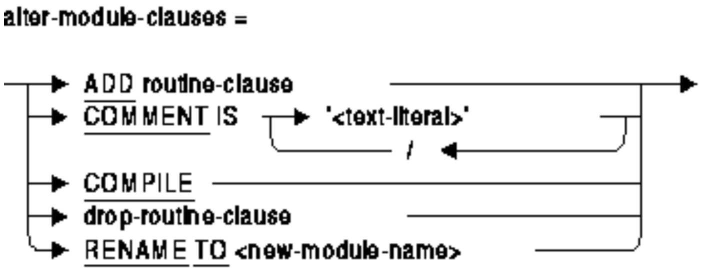
Environment

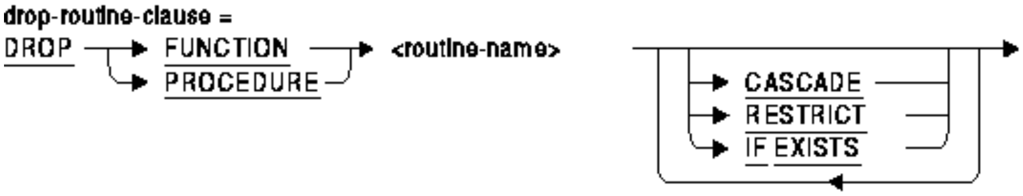
You can use the ALTER MODULE statement:

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

Format







Usage Notes

- You require ALTER privilege on the referenced module.
- The ALTER MODULE statement causes the RDB$LAST_ALTERED column of the RDB$MODULES table for the named module to be updated with the transactions timestamp.
- The COMPILE option forces the Rdb server to recompile all the stored (SQL) functions and procedures. External routines in the module are not affected.
  Use COMPILE when a routine has been made INVALID by the execution of a DROP ... CASCADE operation. This mechanism should be preferred over the SET FLAGS 'VALIDATE_ROUTINE' method available in previous versions.
  The first routine which cannot successfully be compiled will be reported and the ALTER statement terminated.
- The ADD clause allows new functions and procedures to be added to the module. Please refer to the CREATE MODULE statement for details. The END MODULE clause must be used to end the ALTER MODULE clause to provide an unambiguous statement termination.
- RENAME TO allows the name of the module to be changed. This clause requires that synonyms are enabled in the database. Use ALTER DATABASE ... SYNONYMS ARE ENABLED.
  The old name will be used to create a synonym for the new name of this module. This synonym can be dropped if the name is no longer used by applications.
  This clause is equivalent to the RENAME MODULE statement.
- The DROP FUNCTION and DROP PROCEDURE clauses will drop the named routines from this module. All DROP clauses are executed prior to the COMPILE and ADD clauses in this ALTER statement. The named function or procedure must be part of the module being altered.
- The COMMENT IS clause changes the existing comment on the module. This clause is equivalent to the COMMENT ON MODULE statement.

Note

In this release, no LOCAL TEMPORARY TABLE definitions for the module are visible to the ADD FUNCTION or ADD PROCEDURE clauses. This restriction will be lifted in a

future release of Rdb.

Examples

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

Example 9−1 Changing the comment on a module

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

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

 Routines in module EMPLOYEES_MAINTENANCE:
     ADD_EMPLOYEE
     IS_CURRENT_EMPLOYEE
     REMOVE_EMPLOYEE
```

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

Example 9−2 Revalidating all routines in a module

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

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

Example 9−3 Replacing a routine in a module

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

 Header:
 EMPLOYEES_MAINTENANCE
 Module ID is: 7

 Routines in module EMPLOYEES_MAINTENANCE:
     ADD_EMPLOYEE
     REMOVE_EMPLOYEE

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

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

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

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

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

 Routines in module EMPLOYEES_MAINTENANCE:
     ADD_EMPLOYEE
     REMOVE_EMPLOYEE
```

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

Example 9−4 Adding a new function to a module

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

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

 Routines in module EMPLOYEES_MAINTENANCE:
     ADD_EMPLOYEE
     IS_CURRENT_EMPLOYEE
     REMOVE_EMPLOYEE
```

9.3.18 New RENAME Statement

Description

This release of Oracle Rdb 7.1 includes a new RENAME statement and support for the RENAME TO clause for most ALTER statements.

The RENAME statement allows the database administrator to change the name of a database object. This new name is then available for reference in other data definition statements, as well as from queries and routines.

Note

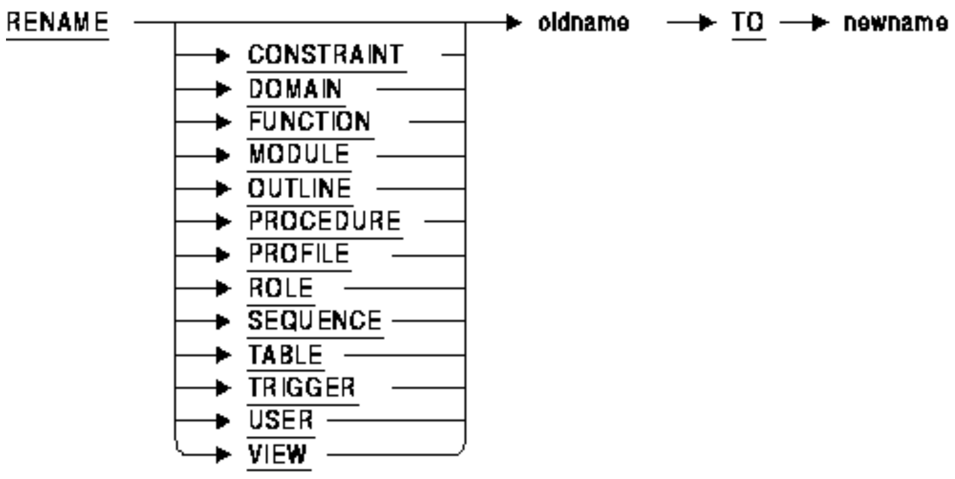
The RENAME statement may require that synonyms are enabled for the database. Please reference the SYNONYMS ARE ENABLED clause of the ALTER, CREATE and IMPORT DATABASE statements.

Environment

You can use the RENAME statement:

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

Syntax



oldname is the name of an existing object in the database. If the object type keyword is specified then an object must exist of that type. The name may also be a synonym for an object of the specified type.

newname is the new name for this object. This name must not already exist in the database for this object type, nor be the name of a synonym. The one exception is when the synonym references the ***oldname*** object. See the usage notes for further discussion.

If this is a RENAME TABLE, RENAME VIEW or RENAME SEQUENCE then the ***newname*** cannot be the name of an existing table, sequence or view.

USAGE NOTES

- You must have ALTER privilege on the database to rename a DOMAIN or OUTLINE.
  You must have ALTER privilege on the table, view, sequence, module, function or procedure to alter its name. If the procedure or function is part of a module, then you will require only ALTER privilege on the containing module.
  You must have SECURITY privilege on the database to alter the name of a USER, ROLE or PROFILE.
  You must have ALTER privilege on the referencing table to rename a CONSTRAINT or TRIGGER.
- The names of the database objects are stored in the Rdb system tables as both column values (for instance RDB$SEQUENCE_NAME) as well as encoded in binary definitions (such as RDB$VIEW_RSE and RDB$VIEW_SOURCE).
  The RENAME clause will modify all column values to reference the new name. However, the encoded values and original SQL source code are not modified by this command.
  To support these encoded definitions, as well as previously compiled applications, the old names are

used to create synonyms that reference the new name of the object.
The RENAME statement will create a synonym for the old names of domains, functions, modules, procedures, sequences, tables and views. These synonyms can be dropped if they are not used.

Note

It is not possible to create synonyms for OUTLINES, CONSTRAINTS, PROFILES, ROLES, TRIGGERS, or USERS. Therefore, RENAME does not create synonyms for these objects. Care should be taken if the old names appear in module definitions or application code.

- If a synonym already exists and references the same object, then it will be removed as part of the RENAME statement. For example, if you rename a table and wish to return to the previous ***oldname***, there will be an existing synonym with this name. Rdb will implicitly remove this synonym during the rename operation.
- The object type is optional. If no object type keyword is provided then Rdb will search for a matching name in this order:
    1. table or view
    2. domains
    3. function or procedure
    4. module
    5. sequence
    6. trigger
    7. constraint
    8. outline
    9. user
    10. role
    11. profile
- When an IDENTITY column is created for a table, a sequence with the same name as the table is implicitly created. You may not use RENAME SEQUENCE on the identity sequence. Please use RENAME TABLE instead to alter the name of the table and its identity sequence.
- You may not RENAME an Rdb system table, system view or system sequence.
- The following table compares the ALTER commands to the equivalent RENAME statement:

Table 9−1 Comparison between RENAME and ALTER statements

RENAME statement	Equivalent ALTER statement	Is a synonym created?
RENAME CONSTRAINT	ALTER CONSTRAINT ... RENAME TO	No
RENAME DOMAIN	ALTER DOMAIN ... RENAME TO	Yes
RENAME FUNCTION	ALTER FUNCTION ... RENAME TO	Yes
RENAME MODULE	ALTER MODULE ... RENAME TO	Yes
RENAME OUTLINE	ALTER OUTLINE ... RENAME TO	No
RENAME PROCEDURE	ALTER PROCEDURE ... RENAME TO	Yes
RENAME PROFILE	ALTER PROFILE ... RENAME TO	No
RENAME ROLE	ALTER ROLE ... RENAME TO	No
RENAME SEQUENCE	ALTER SEQUENCE ... RENAME TO	Yes

RENAME TABLE	ALTER TABLE ... RENAME TO	Yes
RENAME TRIGGER	ALTER TRIGGER ... RENAME TO	No
RENAME USER	ALTER USER ... RENAME TO	No
RENAME VIEW	No equivalent	Yes

9.3.19 New Warning Generated When Row Size Exceeds Row Cache Length

Bug 2909840

When a table's row length exceeds the size allocated for the associated logical area row cache, Rdb generates a warning to alert the database administrator of a possible problem with the row cache.

Rows can change size when ALTER TABLE adds a new column or alters an existing column's data type. An existing column can also be implicitly altered by ALTER DOMAIN, as the domain change is propagated to all referencing tables.

The following example shows the new warning message.

```
SQL> alter table NEW_EMPLOYEES alter column MIDDLE_INITIAL varchar(20);
%RDB−W−META_WARN, metadata successfully updated with the reported warning
−RDMS−I−RCHLENEXC, new row length exceeds size of row cache − check cache
attributes
```

Note that this ALTER TABLE operation succeeded. The warning is generated because rows that are too long to fit in the cache will not benefit from fast in−memory access. However, it is also quite possible that row compression will continue to result in compressed rows that still fit within the row cache. The database administrator will need to evaluate the change in light of the database design.

9.3.20 Oracle Media Management V2.0 API for Oracle Rdb RMU

The Oracle Media Management V2.0 API is supported by Oracle Rdb for RMU commands which accept the /LIBRARIAN qualifier. This support permits backing up to and restoring from, data archiving software applications supporting this interface. Examples of such applications include:

- ♦ Archive Backup System for OpenVMS from Hewlett−Packard Corporation on the world−wide−web at *http://h71000.www7.hp.com/openvms/storage/abspage.html*
- ♦ LEGATO NetWorker(R) from LEGATO Systems, Inc. on the world−wide−web at *http://www.legato.com/*
- ♦ Archive Backup Client (ABC) for OpenVMS from STORServer Inc. on the world−wide−web at *http://www.storserver.com/*

More information on these products is available from the vendors.

9.3.20.1 Commands Accepting /LIBRARIAN

The Oracle Rdb Release 7.1.2 RMU commands which accept the /LIBRARIAN qualifier for storing data to this interface are:

- RMU /BACKUP
- RMU /BACKUP /AFTER_JOURNAL
- RMU /OPTIMIZE /AFTER_JOURNAL

The Oracle Rdb Release 7.1.2 RMU commands which accept the /LIBRARIAN qualifier for retrieving data from this interface are:

- RMU /RESTORE
- RMU /RESTORE /ONLY_ROOT
- RMU /DUMP /AFTER_JOURNAL
- RMU /DUMP /BACKUP_FILE
- RMU /RECOVER

RMU only supports the retrieval using the /LIBRARIAN qualifier for data that has been previously stored by RMU using the /LIBRARIAN qualifier.

In addition to the /LIBRARIAN qualifier used with existing RMU commands, there is a new RMU /LIBRARIAN /LIST command to list data streams stored in a LIBRARIAN implementation that have been created by RMU from a backup filename and a new RMU /LIBRARIAN /REMOVE command to delete data streams stored in a LIBRARIAN implementation that have been created by RMU from a backup filename.

Oracle Media Manager V2.0 Interface

Only applications that conform to the Oracle Media Manager V2.0 specification can be called using the /LIBRARIAN qualifier or the new RMU /LIBRARIAN commands.

RMU commands used with the /LIBRARIAN qualifier may not specify a list of tape or disk devices. It accepts a backup file ("rbf file") name. Any disk or device specification and version number specified with the backup file name is ignored for the backup file name specified to the archive. For example, "device:[directory]FILENAME.RBF;1" is specified as "FILENAME.RBF " when the backup file data is stored in or retrieved from the archive.

9.3.20.2 Opaque Archive Application

The archive application is effectively an opaque "black box" in regards to RMU and the backup file name is the identifier of the stream of data stored in the archive. The utilities and command procedures specific to the particular LIBRARIAN application must be used to associate devices with the stream of data sent to or retrieved from the archive by RMU. Since the LIBRARIAN application is an opaque utility to RMU that stores and manages data, device specific qualifiers such as /REWIND, /DENSITY or /LABEL cannot be used with this interface.

9.3.20.3 RMU Backup Streams

Each writer thread for a backup operation or reader thread for a restore operation manages its own stream of data. Therefore, each thread uses a unique backup file name generated from the backup file name specified on the command line. A number is incremented and added to the end of the backup file extension (the extension defaults to ".RBF") specified to the archive (except for the first) representing a unique data stream. This number is the equivalent of the volume number associated with non−LIBRARIAN RMU backups and restores.

For example, if

```
$RMU /BACKUP /LIBRARIAN=(WRITER_THREADS=3) /LOG DB FILENAM.RBF
```

is specified for a backup command, the backup file data stream names

```
FILENAME.RBF
FILENAME.RBF02
FILENAME.RBF03
```

are specified to the archive to identify the three streams of data stored in the archive by the three writer threads which together represent the stored database. Since each data stream must contain at least one database storage area and a single storage area must be completely contained in one data stream, if the number of writer threads specified is greater than the number of storage areas, it will be set equal to the number of storage areas.

When

```
$RMU /RESTORE /LIBRARIAN=(READER_THREADS=3) /LOG FILENAM.RBF
```

is specified to restore the database, these same three data stream backup file names, one name specified by each of the three reader threads, will be generated by RMU and sent to the archive application to retrieve all the data associated with the database. If the number of reader threads is less than the number of backup writer threads, one or more restore reader threads will restore more than one data stream. If the number of reader threads specified is greater than the number of backup writer threads, the number of reader threads will be set equal to the number of backup writer threads so that all restored data is retrieved.

Therefore, the same number of reader threads in the example was specified on the restore as writer threads on the backup to generate all the stream names which represent the database. The user does not have to specify the same number of reader threads on the restore as writer threads specified on the backup. If a smaller number of reader threads on the restore is specified than the number of writer threads specified in the backup of the database, the data streams to be retrieved will be divided among the specified reader threads using an algorithm which assigns the data streams so that each thread will have an approximately equal amount of work to do. If a greater number of reader threads is specified on the restore than was specified on the backup, the number of reader threads will be automatically changed to equal the number of writer threads used in the backup.

The /VOLUMES qualifier cannot be used on the RMU/RESTORE command if the /LIBRARIAN qualifier is used. RMU automatically determines the number of data streams stored in the LIBRARIAN implementation based on the backup file name specified for the restore command and sets the volume number to the actual number of stored data streams. This helps to ensure that all data streams which represent the database are retrieved.

The default value for both WRITER_THREADS and READER_THREADS is "1". The WRITER_THREADS parameter can only be specified with the /LIBRARIAN qualifier for the RMU/BACKUP database command. The READER_THREADS parameter can only be specified with the /LIBRARIAN qualifier for the RMU /RESTORE database and the RMU /DUMP /BACKUP commands. All other RMU commands that accept the /LIBRARIAN qualifier only use one writer thread or one reader thread representing one archive data stream.

9.3.20.4 Parallel Backup Operations

The /LIBRARIAN qualifier can be used for parallel backup operations where backup threads can execute in multiple processes distributed among one or more nodes in a cluster. The database backup command can be invoked as a parallel command which uses multiple processes but the other RMU commands which accept the /LIBRARIAN qualifier do not support parallel processes but execute in one process.

The following lines in the backup PLAN file used to specify the parameters for parallel backup operations relate directly to the LIBRARIAN feature.

```
    Backup File = MF_PERSONNEL.RBF
    Style = Librarian
    Librarian_trace_level = #
    Librarian_trace_file = FILE.TRACE
    Librarian_logical_names = (−
             logical_name_1=equivalence_value_1, −
             logical_name_2=equivalence_value_2)
    Writer_threads = #
```

The backup file name must be the same file name specified for the restore and the style must be set to "Librarian" indicating a backup to the LIBRARIAN. The "Librarian_logical_names" entry is a list of logical names and their equivalence values. This is an optional parameter provided so that any logical names used by a particular LIBRARIAN application can be defined as process logical names before the backup or restore operation begins. For example, some LIBRARIAN applications provide support for logical names for specifying catalogs or debugging. "Librarian_trace_level = #" and "Librarian_trace_file = FILE.TRACE" are optional parameters specified with the /LIBRARIAN qualifier and passed to the LIBRARIAN application to be used for diagnostic purposes. The "Writer_threads = #" specifies the number of writer threads which will be used by each worker executor process. If this number exceeds the number of database storage areas assigned to a worker process, it will be set equal to the number of storage areas specified for that worker process.

If the backup is a parallel operation, a PLAN file is created and executed as part of the existing RMU /BACKUP /PARALLEL and RMU /BACKUP /PLAN command syntax. The following is an example of a parallel backup and non−parallel restore (the restore is always non−parallel and executes in a single process) using the /LIBRARIAN qualifier.

```
$RMU /BACKUP /PARALLEL=EXECUTOR=3 /LIBRARIAN=WRITER_THREADS=3−
 /LIST_PLAN=FILENAME.PLAN /NOEXECUTE /LOG DATABASE FILENAM.RBF
$RMU /BACKUP /PLAN FILENAME.PLAN
$RMU/ RESTORE /LIBRARIAN=(READER_THREADS=9) /LOG FILENAME.RBF
```

In this example, the first backup command creates the PLAN file for a parallel backup but does not execute it. The second backup command executes the parallel backup using the PLAN file. Note that 3 worker processes will be used and each process will use the 3 writer threads specified with the /LIBRARIAN qualifier. Each writer thread in each process will write one stream of backup data to the LIBRARIAN. Therefore 9 streams will be written to the LIBRARIAN archive. The streams will be given the names:

```
FILENAME.RBF
FILENAME.RBF02
FILENAME.RBF03
FILENAME.RBF04
FILENAME.RBF05
FILENAME.RBF06
FILENAME.RBF07
FILENAME.RBF08
FILENAME.RBF09
```

To retrieve the same 9 data streams which represent the backed up Rdb database on the non−parallel restore, a READER_THREADS=9 parameter can be specified with the /LIBRARIAN qualifier to use 9 threads to execute the restore, or if a READER_THREADS value between 1 and 8 is specified (1 is the default), RMU will determine the number of data streams actually stored by querying the LIBRARIAN implementation and distribute the data streams among the requested reader threads. If a READER_THREADS value is specified that is greater than "9", RMU will set it to "9" so that the restore does not attempt to retrieve data streams which do not exist.

9.3.20.5 Data Stream Naming Considerations

Since data stream names representing the database are generated based on the backup file name specified for the RMU backup command used with the /LIBRARIAN qualifier, the user must either use a different backup file name to store the next backup of the database to the LIBRARIAN implementation or first delete the existing data streams generated from the backup file name before the SAME backup file name can be reused for the next backup.

To delete the existing data streams stored in the LIBRARIAN implementation, a LIBRARIAN management utility can be used or the RMU /LIBRARIAN /REMOVE command described below can be used with just the backup file name to delete all the data streams generated based on that name. The user can incorporate the date or some other unique identifier in the backup file name when he does each backup to make it unique if he wants to avoid deleting a previous backup to the LIBRARIAN which used the same backup file name. Many LIBRARIAN implementations allow the user to specify an automatic deletion date for each data stream stored in their archives.

9.3.20.6 /LIBRARIAN Parameters

The /LIBRARIAN qualifier accepts the following parameters.

- ♦ WRITER_THREADS=#
  Use # writer threads to write # backup data streams to the LIBRARIAN. The database storage areas will be partitioned among the database streams. The streams will be named BACKUP_FILENAME.EXT, BACKUP_FILENAME.EXT02, BACKUP_FILENAME.EXT03, up to BACKUP_FILENAME.EXT99. BACKUP_FILENAME.EXT is the backup file name specified in the RMU command excluding any specified device, directory or version number. The default extension name is ".RBF". The "WRITER_THREADS" parameter can only be specified for parallel and non−parallel database backups. The default is 1 writer thread. The minimum is 1 and the maximum is 99. This parameter cannot be specified for other RMU commands which accept the /LIBRARIAN qualifier for write operations such as RMU/BACKUP/AFTER_JOURNAL/LIBRARIAN since these commands only allow 1 writer thread which creates 1 database stream. This value will be set equal to the number of database storage areas if it exceeds that number.

- ♦ READER_THREADS=#
  Use # reader threads to read all the backup data streams from the LIBRARIAN created for the backup filename. The streams will be named BACKUP_FILENAME.EXT, BACKUP_FILENAME.EXT02, BACKUP_FILENAME.EXT03, up to BACKUP_FILENAME.EXT99. BACKUP_FILENAME.EXT is the backup file name specified in the RMU command excluding any specified device, directory or version number. The default extension name is ".RBF". The "READER_THREADS" parameter can only be specified for database restores and dumps of databases stored by RMU in the LIBRARIAN. A reader thread value of 1 is used for all other RMU commands that read data from the LIBRARIAN. The minimum READER_THREADS value is 1 and the maximum is 99. The default value is 1.
  The number of READER_THREADS for a database restore from the LIBRARIAN should be equal to or less than the number of WRITER_THREADS specified for the database backup or the number of reader threads will be set by RMU to be equal to the number of data streams actually stored in the LIBRARIAN by the backup. If the READER_THREADS specified for the restore are less than the WRITER_THREADS specified for the backup, RMU will partition the data streams among the specified reader threads so that all data streams representing the database are restored. Therefore, each reader thread may read more than one data stream.
- ♦ TRACE_FILE=file_specification
  The LIBRARIAN application which supports the MEDIA MANAGEMENT API V2.0 will write trace data to this file, if specified, as defined in the MEDIA MANAGEMENT API V2.0 specification.
- ♦ LEVEL_TRACE=#
  The level number of the trace data written by the LIBRARIAN application which supports the MEDIA MANAGEMENT API V2.0 as defined in the MEDIA MANAGEMENT API V2.0 specification (levels 0 through 2) or a higher level as defined by the LIBRARIAN application. Level 0 (trace all error conditions) is the default.
- ♦ LOGICAL_NAMES=(logical_name=equivalence_value,...)
  This parameter allows the user to specify a list of process logical names which the LIBRARIAN application may use to specify particular catalogs or archives for storing or retrieving backup files, or LIBRARIAN debug logical names, etc. See the LIBRARIAN specific documentation for the definition of these logical names. The list of process logical names will be defined by RMU prior to the start of the backup or restore operation.

9.3.20.7 Logical Names To Access LIBRARIAN Application

The following VMS logical names are for use with a LIBRARIAN application. These logical names need to be defined before the RMU backup or restore command is executed and should not be specified with the list of logical names specified with the /LIBRARIAN qualifier.

- ♦ RMU$LIBRARIAN_PATH
  This logical name must be defined to the file specification for the sharable LIBRARIAN image to be loaded and called by RMU backup and restore operations. The translation must include the file type (".EXE" for example) and must not include a version number. The shareable LIBRARIAN shareable image referenced must be an installed ("known") image. See the LIBRARIAN implementation documentation for the name and location of this image and how it should be installed. For a parallel RMU backup, RMU$LIBRARIAN_PATH should be defined as a system−wide logical name so that the multiple processes created by a parallel backup can all translate the logical.

```
$ DEFINE /SYSTEM /EXECUTIVE_MODE −
    RMU$LIBRARIAN_PATH librarian_shareable_image.exe
```

- RMU$DEBUG_SBT
  This logical name is not required. If defined to any value, RMU will display debug tracing information messages from modules that make calls to the LIBRARIAN shareable image. This information may be helpful for support analysts from Oracle or your librarian vendor when analyzing problems. See the LIBRARIAN documentation for any other logical names or setup procedures specific to the particular LIBRARIAN implementation. For a parallel backup, RMU$DEBUG_SBT should be defined as a system logical so that the multiple processes created by a parallel backup can all translate the logical.

9.3.20.8 SQL/Services Required for RMU Parallel Backup

Oracle Rdb V7.1 SQL/Services is required for RMU parallel backup operations. However, no special changes are required to SQL/Services specific to RMU parallel backup operations to the LIBRARIAN. The LIBRARIAN should be installed and available on all nodes on which the parallel backup operation executes. As long as non−LIBRARIAN Rdb V7.1 parallel backup operations are currently working, no LIBRARIAN specific changes to the SQL/Services setup should be needed.

9.3.20.9 Listing and Deleting Data Streams

The RMU /LIBRARIAN command enables the user to list or delete data streams stored in the LIBRARIAN implementation based on the backup file name used for the RMU backup. The LIST and REMOVE options cannot be used together in the same RMU/LIBRARIAN command.

```
RMU /LIBRARIAN /LIST=(OUTPUT=disk:[directory]listfile.ext) FILENAME.RBF
RMU /LIBRARIAN /REMOVE=([NO]CONFIRM) FILENAME.RBF
```

FILENAME.RBF is the backup filename. Any device, directory or version number specified with the backup file name will be ignored. The backup file name must be the same name previously used for an RMU backup to the LIBRARIAN. A default file type of ".RBF" is assumed if none is specified.

The following command qualifiers are supported:

- /LIST=(OUTPUT=disk:[directory]listfile.ext)
  "/LIST" used alone will display to the default output device. If the "OUTPUT" option is used, output will be displayed to the specified file. All data streams existing in the LIBRARIAN that were generated for the specified backup name will be listed. The information listed for each data stream name include:
    - The backup stream name based on the backup file.
    - Any comment associated with the backup stream name.
    - The creation method associated with the backup stream name. This will always be STREAM to indicate creation by a backup operation.
    - The creation date and time when the stream was backed up to the LIBRARIAN.
    - Any expiration data and time specified for deletion of the stream by the LIBRARIAN.
    - The media sharing mode which indicates if the media can be accessed concurrently or not. This is usually the case for disks but not tapes.
    - The file ordering mode which indicates if files on the media can be accessed in random order or sequential order.
    - Any volume label(s) for the media which contain the backup stream.

Implementation Specific

Not all of these items will be listed depending on the particular LIBRARIAN implementation.

- /REMOVE=([NO]CONFIRM)
  "/REMOVE" deletes all data streams existing in the LIBRARIAN that were generated for the specified backup name. This command should be used with caution. The user should be sure that a more recent backup for the database exists in the LIBRARIAN under another name before using this command. The "CONFIRM" option is the default. It will prompt the user to confirm that he wants to delete the backup from the LIBRARIAN. The user can then reply "Y(ES)" to do the deletion or "N(O)" to exit the command without doing the deletion if he wants to confirm that a more recent backup for the database exists in the LIBRARIAN that was generated using a different backup name. The user must specify the "NOCONFIRM" option if he does not want to be prompted. In this case, the deletion will be done with no confirmation prompt.
  The following additional optional keywords can be specified with either the /LIST qualifier or the /REMOVE qualifier. They must be specified and have no defaults. These are the same options discussed earlier for the /LIBRARIAN qualifier used with other RMU commands such as /BACKUP and /RESTORE.
  - TRACE_FILE=file_specification
    The LIBRARIAN application which supports the MEDIA MANAGEMENT API V2.0 will write trace data to this file, if specified, as defined in the MEDIA MANAGEMENT API V2.0 specification.
  - LEVEL_TRACE=n
    The level number of the trace data written by the LIBRARIAN application which supports the MEDIA MANAGEMENT API V2.0 as defined in the MEDIA MANAGEMENT API V2.0 specification (levels 0 through 2) or a higher level as defined by the LIBRARIAN application. Level 0 (trace all error conditions) is the default.
  - LOGICAL_NAMES=(logical_name=equivalence_value,...)
    This parameter allows the user to specify a list of process logical names which the LIBRARIAN application may use to specify particular catalogs or archives for listing or removing backup files, or LIBRARIAN debug logical names, etc. See the LIBRARIAN specific documentation for the definition of these logical names. The list of process logical names will be defined by RMU prior to the start of the list or remove operation.

9.3.21 Sanity Checks Added to RMU /VERIFY to Check TSNs and CSNs

Bug 2551131

Checks have been added to RMU /VERIFY to ascertain if the Transaction Sequence Numbers (TSNs) and Commit Sequence Numbers (CSN) have acceptable values. This verification is performed while verifying the root.

Three sanity checks have been added to achieve this. They are:

- Highest Active TSN is less than the TSN that will be assigned to the next transaction. If this check fails the following kind of error message is displayed.

  ```
  %RMU−E−HIGHTSNINV, Highest active TSN (1024:1024) is higher than
  the TSN that will be assigned next (0:256).
  ```
- Last Committed TSN is less than the TSN that will be assigned to the next transaction. If this check fails the following kind of error message is displayed.

  ```
  %RMU−E−LASTCMTSNINV, Last Committed TSN (1024:1024) is higher than
  the TSN that will be assigned next (0:256).
  ```
- Highest CSN is less than the next CSN that will be assigned. If this check fails the following kind of error message is displayed.

  ```
  %RMU−E−HIGHCSNINV, Highest CSN (1024:1024) is higher than the CSN
  that will be  assigned next (0:256).
  ```

These checks will be available starting with Oracle Rdb Release 7.1.2.

9.3.22 RMU /CLOSE /WAIT /NOCLUSTER Now Allowed

Bug 976101

In the past, if the /WAIT qualifier was used with ***RMU /CLOSE*** then it implied ***/CLUSTER***. Specifying ***/NOCLUSTER*** was not allowed. This restriction has been lifted. It is now possible to specify ***/NOCLUSTER*** in conjunction with the /WAIT qualifier. This provides the ability to have database shutdown complete on the local node before RMU returns to the DCL prompt. Specifying the /WAIT qualifier without the ***/NOCLUSTER*** qualifier will still imply ***/CLUSTER***, as it has in prior releases.

9.3.23 Native 64−bit Virtual Addressing for Row Caches

Oracle Rdb Release 7.1.2 provides enhancements to the Row Cache feature to utilize the native 64−bit virtual memory addressing capabilities of the Alpha processor and OpenVMS. Utilizing these enhancements, applications are now able to more easily access row caches with significantly larger numbers of records being cached.

9.3.23.1 Background

Within Oracle Rdb, the VLM (Very Large Memory, or "LARGE MEMORY IS ENABLED") feature was created circa 1995 for Rdb release 7.0 to allow access to more than 32 bits worth of virtual address space (the traditional VMS address space size). This interface was implemented because, at the time, programs on OpenVMS Alpha did not have the ability to directly access memory outside a 32−bit virtual address space.

In addition, the "SHARED MEMORY IS SYSTEM" feature was implimented to help reduce consumption of process virtual address space in the "program region" (P0 region). By moving database shared memory sections from P0 to "system" (S0/S1) address space, additional program code and buffers could be stored in P0 address space. However, even this additional virtual address space was limited to something less than 2 GB (gigabytes) of usable memory.

Starting with OpenVMS Alpha V7.0, the operating system provides native support for a 64−bit virtual address space, defined by the Alpha architecture. This capability makes 64−bit virtual address space available to applications. OpenVMS and Current Alpha architecture implementations support 8 TB (terabytes) of virtual address space.

Previously, the Oracle Rdb Row Cache feature was limited to something less than 33 million total cached rows per database. This limitation was due primarily to storing row cache related data structures in 32−bit address space. Even with the "LARGE MEMORY IS ENABLED" attribute, some data structures had to be located in 32−bit virtual address space and lead to this restriction.

9.3.23.2 64−bit Addressing

Starting with Oracle Rdb Release 7.1.2, the Row Cache feature utilizes the native 64−bit virtual addressing support within the Alpha processor and OpenVMS. Various data structures related to the Row Cache feature are now created in the OpenVMS "P2" 64−bit virtual address space. The existing Row Cache memory mapping features "LARGE MEMORY IS ENABLED" and "SHARED MEMORY IS SYSTEM" have been eliminated and replaced with this native 64−bit virtual addressing. Row caches on Alpha processors are now always mapped in 64−bit process virtual address space.

9.3.23.3 No Application or Database Changes Required

There should be no user or application visible effects due to the Oracle Rdb implementation of native 64−bit virtual addressing for Row Caches. If the "RESIDENT" attribute is specified for a row cache, the cache will be created as a memory−resident global section utilizing OpenVMS "shared page tables" (potentially with granularity hints). The OpenVMS "shared page tables" capability for memory−resident global sections can help reduce physical memory consumption by allowing multiple processes to share page table entries for the global section. For very large global section, "granularity hints" allow ranges of pages to be mapped by a single translation buffer entry within the Alpha processor leading to improved translation buffer hit rates.

When the shared memory section for a row cache is to be memory−resident, the pages for the section are always resident in physical memory. The pages are not placed into the process' working set list when the process maps to the global section. The pages are also not charged against the process' working set quota or against any page−file quota. Pages within memory−resident global sections are not backed by the pagefile or by any other file on disk. The user must have the rights identifier VMS$MEM_RESIDENT_USER to create the memory−resident global section.

9.3.23.4 Deprecated Attributes

The row cache memory mapping features "LARGE MEMORY IS ENABLED" and "SHARED MEMORY IS SYSTEM" have been replaced with the "RESIDENT" attribute. If the "LARGE MEMORY IS ENABLED" or "SHARED MEMORY IS SYSTEM" attributes are specified for a row cache, the cache is considered to be set "RESIDENT". Though deprecated, the "LARGE MEMORY IS ENABLED" and "SHARED MEMORY IS SYSTEM" attributes are internally considered as synonyms for "RESIDENT".

The following Oracle Rdb Row Cache attributes have become deprecated in SQL:

- "LARGE MEMORY IS ENABLED"
- "SHARED MEMORY IS SYSTEM"

- ♦ "WINDOW COUNT"

The following Oracle Rdb Row Cache attribute keywords have become deprecated with the "RMU /SET ROW_CACHE /ALTER" command:

- ♦ WINDOW_COUNT=n
- ♦ SHARED_MEMORY=TYPE=SYSTEM

The "LARGE MEMORY IS ENABLED" and "SHARED MEMORY IS SYSTEM" attributes remain supported and functional for database and global buffer shared memory configuration.

9.3.23.5 Cache Size Limits

In this release of Oracle Rdb, the total number of rows for any individual cache (the combination of "live" rows and snapshot rows) is limited to 2,147,483,647. This restriction may be relaxed in a future Oracle Rdb release.

9.3.23.6 Row Cache Feature Only

This change to utilize native 64−bit virtual addressing is specific to the Row Cache feature in this release of Oracle Rdb. Rdb database global buffers, for example, continue to be mapped into 32−bit virtual address space and continue to support the "LARGE MEMORY IS ENABLED" and "SHARED MEMORY IS SYSTEM" attributes. No additional support for 64−bit virtual addressing (including via callable interfaces such as SQL) is provided in this release of Oracle Rdb. Oracle is considering additional possible uses for native 64−bit virtual addressing within Oracle Rdb for future releases.

9.3.23.7 System Parameters

Shared memory sections using the "LARGE MEMORY IS ENABLED" or "SHARED MEMORY IS SYSTEM" features were previously not created as OpenVMS global sections and were not directly effected by the global section system parameters (specifically GBLSECTIONS, GBLPAGES and GBLPAGFIL). Because all row cache shared memory sections are now global sections on OpenVMS, it is possible that the global section system parameters may have to be increased in order to map large caches that previously relied on the "LARGE MEMORY IS ENABLED" or "SHARED MEMORY IS SYSTEM" features.

Reserved Memory Registry

The Reserved Memory Registry allows an OpenVMS system to be configured with large amounts of memory set aside for use within memory−resident sections or other privileged code. This release of Oracle Rdb does not support use of the OpenVMS Reserved Memory Registry for registering Oracle Rdb shared memory sections. This restriction may be relaxed in a future Oracle Rdb release.

9.3.23.8 Additional Information

Refer to the following documents for additional information about Alpha processors and OpenVMS addressing and memory management:

- ♦ OpenVMS Programming Concepts Manual
- ♦ OpenVMS System Services Reference Manual
- ♦ OpenVMS Calling Standard
- ♦ OpenVMS System Manager's Manual
- ♦ OpenVMS Alpha Partitioning and Galaxy Guide
- ♦ Alpha Architecture Handbook

9.3.24 Snapshots In Row Cache

Oracle Rdb Release 7.1.2 provides enhancements to the Row Cache feature to allow snapshot copies of rows to be stored in Row Cache shared memory rather than in the on−disk snapshot storage areas. Utilizing these enhancements, applications are now able to more easily access and modify rows with reduced database I/O and locking operations. These features are enabled and configured on a per−cache basis.

9.3.24.1 Background

A row cache is a section of globally accessible memory that contains copies of rows. Row caching provides the ability to store frequently accessed rows in memory, reducing disk I/O. The rows remain in memory even when the associated page has been flushed back to disk. A row cache can contain index structures as well as table data.

The snapshot mechanism in Oracle Rdb allows read−only transactions to see a consistent view of the database while other transactions update the database. The previous versions of rows are written to special snapshot areas of the database by the transactions that update the rows.

By default, snapshot copies of rows are stored in database snapshot storage area files. The "Snapshots in Row Cache" feature allows snapshot rows to be stored in a designated section of shared memory. With this enhancement, read−only transactions can quickly read snapshot copies of rows from memory and read−write transactions can quickly write snapshots. The reading and writing of the snapshot information can be accomplished with no database page I/O or associated database page locking.

9.3.24.2 Configuration

Each defined row cache for a database can be designated to allow a specified number of snapshot rows to be stored in the cache. The number of snapshot rows allowed is specified in addition to the number of database rows that can be stored in the cache. Because many versions of a row may be stored in the snapshot portion of the cache, the number of snapshot rows and cache rows are specified independently.

As the snapshot portion of the row cache is effectively an extension of the row cache itself, most attributes of the cache are shared with the snapshot portion. These attributes include:

- ♦ The maximum row size (as is specified with the ROW LENGTH is n BYTES parameter)
- ♦ Memory location specification (SHARED MEMORY IS PROCESS, SHARED MEMORY IS SYSTEM, LARGE MEMORY, and so on)
- ♦ Allocate Set count

9.3.24.3 Space Reclaimation

A snapshot copy of a database record must be maintained until there are no transactions in the database older than the transaction that stored the snapshot copy of the record. As the oldest transactions in the database commit, space used to store snapshot records (either in the snapshot storage areas or in the snapshot portion of a cache) may be re−used for storing newer snapshots. By keeping transactions relatively short, the number of snapshot rows that need to be stored can be reduced.

9.3.24.4 Objects in Mixed Format Areas

With this release of the "Snapshots in Row Cache" feature, only objects (index nodes and data records) stored in uniform−format storage areas utilize the ability to store snapshots in a row cache. Objects stored in mixed format storage areas are unable to have snapshot copies stored in a row cache.

This restriction results from internal mechanisms used to perform sequential scans in the database and interactions with the retrieval of snapshot information. This restriction is expected to be relaxed in a future Oracle Rdb release.

9.3.24.5 SQL Syntax

The SQL "ALTER DATABASE ... ALTER ROW CACHE", "ALTER DATABASE ... ADD ROW CACHE", and " CREATE DATABASE ... CREATE ROW CACHE" commands can be used to set parameters for the snapshot portion of a row cache. The syntax to support snapshots in cache is "ROW SNAPSHOT [IS] { DISABLED | { ENABLED [(CACHE SIZE IS N ROWS)] }".

- ♦ The "ROW SNAPSHOT IS DISABLED" option disables storing snapshot copies of rows within the cache.
- ♦ The "ROW SNAPSHOT IS ENABLED (CACHE SIZE IS n ROWS)" option enables storage of snapshot copies of rows within the cache and specifies the number of snapshot "slots" to allocate for the cache.

If you do not specify the CACHE SIZE clause for the ROW SNAPSHOT IS ENABLED option, Oracle Rdb creates a cache that can contain up to 1000 snapshot rows.

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

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

9.3.24.6 RMU Syntax

The "RMU /SET ROW_CACHE" command can be used to set parameters for the snapshot portion of a row cache. The "/ALTER=(...)" qualifier accepts a "SNAPSHOT_SLOT_COUNT=n" keyword. The value specified on the SNAPSHOT_SLOT_COUNT keyword sets the number of snapshot slots in the cache. A value of zero disables the snapshot portion for the specified cache.

The following example modifies the database MYDB to set the snapshot slot count for the cache "EMPL_IDX" to 25000 slots and disables snapshots in cache for the "SALES" cache:

```
$ RMU /SET ROW_CACHE DGA0:[DB]MYDB.RDB −
  /ALTER=(NAME=EMPL_IDX, SNAPSHOT_SLOT_COUNT=25000) −
  /ALTER=(NAME=SALES, SNAPSHOT_SLOT_COUNT=0)
```

9.3.24.7 Snapshot Cache Sizing

Because the application and workload behaviour determine the number of database rows that are modified and the transaction length, it is not reasonable to make specific recommendations for sizing the snapshot portion of caches for all application and database types. The ratio of the size of the snapshot cache to the main cache may be similar to the ratio of the database snapshot storage area to the live storage area.

The snapshot portion of a row cache may be larger (may contain more rows) than the "main" row cache itself. The snapshot portion of a row cache may also be much smaller.

If an application has long running transactions and active read−write transactions modifying data, many snapshot copies of the modified data may need to be maintained. This can require caches with many snapshot rows for those caches with heavy update activity.

When the snapshot portion of a cache fills and no slots are available for re−use (due to the age of the oldest transaction in the database), read−write transactions may need to "overflow" snapshot records from cache to disk. This overflow operation can be quite costly in terms of CPU time and disk I/O operations. When a snapshot cache is discovered to be full and a read−write transaction must store a new snapshot copy of a row, all existing snapshot copies for that row must be written from the cache to the snapshot storage area on disk. And all future snapshot operations to the cache must also be written to disk.

When the snapshot portion of a row cache is marked as being "full", the row cache server (RCS) process periodically checks the cache to see if space is available for reuse. Similar to the algorithms governing space recollection in snapshot storage areas, this space only becomes available when the oldest transaction in the database commits. When the RCS process finds reclaimable space in the snapshot portion of a cache that is marked "full", it will clear the "full" indicator to permit new snapshot copies to be stored in the cache by read−write transactions.

9.3.24.8 Performance and Operational Considerations

When a row is removed from the cache (due to it becoming fragmented, growing too large for the cache, or from a TRUNCATE TABLE operation), all snapshot copies for the row must be written from cache back to the snapshot storage area on disk. This can be a relatively costly and slow operation. This can usually be avoided by insuring that caches are sized with slots large enough for the data being stored.

At certain times during normal database operations, all modified rows must be written from cache(s) back to the physical database storage areas. Events that require writing all modified data back to the database include:

- Database close
- RMU /VERIFY
- RMU /BACKUP

Prior to "Snapshots in Row Cache", it was unlikely that very many modified rows would remain in cache memory when database snapshots were enabled. Now, for those caches configured with snapshots in cache, the cache itself may have many more modified rows. When there are many modified rows in memory, it may take a significant amount of time to write all of these rows back to the database. You may need to plan based on the amount of time required to, for example, initiate backup operations, if there may be a large number of modified rows in cache.

By removing delays introduced by disk I/O operations, applications may tend to experience improved performance. Some systems, however, may see a significant reduction in system idle time due to the reduction in I/O waiting. Presumably, this will be reflected in an increase of overall application performance as the computer system is now being more effectively utilized.

9.3.24.9 Statistics

The "Row Cache Status" display of the RMU/SHOW STATISTICS utility provides information about the state of a single row cache and now includes status information about the snapshot portion of the cache.

```
Node: CLICK (2/2/2) Oracle Rdb      Perf. Monitor 17−FEB−2003 22:20:39.61
Rate: 3.00 Seconds          Row Cache Status         Elapsed: 00:00:31.79
Page: 1 of 1        DISK$DEMO1:[RDBDEMO]OLTP.RDB;1          Mode: Online
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
                             For Cache: TRD_IDX1
Statistic.Name Stat.Value Percent

Total slots:       2000  100.0% Slot Length: 1000  Hash slots: 2048
Slots full:         876   43.8% Use:        225  25.6%
Slots empty:       1124   56.2% Rsv:        132  11.7%
Marked Slots:       347   17.3% Hot:        347 100.0% Cold:     0   0.0%
Clean Slots:       1653   82.6% Hot:          0   0.0% Cold:  1653 100.0%
Used Space:         876k  43.8% Wstd:         0k  0.0%
Free Space:        1124k  56.2%
Hash Que Lengths:  Empty:1245 1:730      2:73       3:0        4+:0
Cursor position:   1008 of 2000 wrapped 0 times
Cache latched:     No
Cache is full:     No          Cache modified:  Yes  Snapshot is full: No
Number of checkpoints: None
Cache Recovery:    0:3577
Snap Slots:         500  100.0% Ful:      399  79.8% Rcl:     393  78.6%
Snap Cursor:  45 of 500 (slot 2045) wrapped 2 times
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

The following fields provide information about the snapshot portion of the cache.

- Snapshot is full − If snapshots within cache are enabled for the row cache, indicates if the cache has been flagged as having no snapshot slots that can be used to store snapshot records until the oldest transaction in the database commits and allows snapshot slots to be "reclaimed" for re−use.
- Snap Slots − Indicates the number of snapshot slots configured.
- ...Ful − How many of the slots contain snapshot records.
- ...Rcl − How many of the slots contain snapshot records that can be "reclaimed" for re−use.

- Snap Cursor − Indicates the current cursor position within the snapshot portion of the cache; processes allocating slots in the cache start searching for available space at this cursor position.
- ...wrapped − How many times the entire snapshot portion of the cache has been scanned and the allocation cursor was reset to the beginning of the cache.

In this example display, the row cache itself is configured with 2000 slots and the snapshot portion of the cache is configured for 500 slots (the snapshot portion of the cache can be configured to be larger or smaller than the cache itself). The current "Snap Cursor" position is at slot 45 within the 500 snapshot slots.

Within the RMU/SHOW STATISTICS utility, the "Zoom" sub−screen of the "Hot Row Information" display can be used to examine the actual in−cache contents of a row. On this display, the sign (positive or negative) of the "SnapPage" value indicates if the snapshot pointer references a page on disk in the snapshot storage area (a positive value) or a slot number within the snapshot portion of the cache (a negative value). For example, the following display shows a "Zoom" sub−screen for the record in cache with a database key of 56:662:0.

```
Node: MARVEL (1/1/1)      Oracle Rdb  Perf. Monitor  3−FEB−2003 23:11:06.27
Rate: 3.00 Seconds          Hot Row Information        Elapsed: 04:14:58.28
Page: 1 of 6           DGA127:[T]MF_PERSONNEL.RDB;587          Mode: Online
---------------------------------------------------------------------------
                            For Cache: N1 (unsorted)
Area:Page:Ln #Users State Length SlotNo Area:Page:Ln #Users State Length
Empty            0            0      0 Empty             0            0

 +-- DBK=56:662:0 LEN=17 TSN=0:132 SnapPage=−101 VNO=2 -----------------+
 |                                                                       |
 |                     001E  0000  line 0 (56:662:0) record type 30   |
 |                  00 0001  0002  1 byte in 0 sets/dynamic items     |
 |                           ....  12 bytes of static data            |
 |   FC000082020000820100010 2  0005   data '...........ü'            |
 |                                                                       |
 +-----------------------------------------------------------------------+
```

In this example, the snapshot pointer is −101 indicating, because it is negative, that the first entry in the snapshot chain can be found in snapshot slot 101 within the cache. Note that a snapshot pointer of −1 indicates the end of a snapshot chain in this context.

The RMU /DUMP /ROW_CACHE command can also be used to format and display the in−memory cache contents for a row cache of an open database. In this display output, negative snapshot pointer values also indicate snapshot pointers within the row cache.

9.3.24.10 Importance of the After−Image Journal

Any time the Oracle Rdb "Fast Commit" feature is utilized, the after−image journal (AIJ) is the *only* place where changed database records are known to be written to persistent storage when a transaction commits.

Protecting the after−image journal file(s) is thus very important. Oracle encourages use of data protection features such as disk volume shadowing and especially the Oracle Rdb Hot Standby feature to help ensure the safety of the contents of the AIJ.

The Row Cache "backing store" (also known as .RDC) files are also important to ensure rapid database recovery after a system failure. These files contain the modified row content of caches as of the most recent row cache server (RCS) checkpoint operation. Oracle encourages use of data protection features such as disk volume shadowing to help ensure the safety of the contents of the "backing store" files.

9.3.25 Performance Enhancements for RMU /RECOVER with Optimized After−Image Journals

Several enhancements and performance improvements have been made to the creation and processing of optimized after−image journal files. These changes should result in a significant reduction in elapsed time when using optimized after−image journals for recovery.

The RMU /OPTIMIZE /AFTER_JOURNAL command optimizes a backed up after−image journal (.AIJ) file for database recovery (rollforward) operations by eliminating unneeded and duplicate journal records, and by ordering journal records. An optimized after−image journal file created by the RMU /OPTIMIZE /AFTER_JOURNAL command can provide better recovery performance for your database than a non−optimized after−image journal. A potential benefit of this improved recovery performance is that the database is made available to users sooner.

By default, the RMU /OPTIMIZE /AFTER_JOURNAL command orders the after−image journal records by ascending physical DBKEY. The order of records in an optimized AIJ file determines the sequence that pages are accessed by a subsequent RMU /RECOVER command. Sorting AIJ records by physical DBKEY can improve I/O performance at recovery time by reducing disk head motion. However, an ascending physical DBKEY sequence also causes the database to be recovered sequentially, one storage area at a time. This typically results in only one disk device being accessed at a time, often with sequential disk read and write operations.

Significant enhancements have been made to the optimization and usage of optimized after−image journals. These changes include the "/RECOVERY_METHOD" qualifier for the RMU /OPTIMIZE /AFTER_JOURNAL command to allow an alternate record ordering to be used, and asynchronous read−ahead and write−behind for database access during recovery using an optimized after−image journal.

The "/RECOVERY_METHOD" qualifier for the RMU /OPTIMIZE /AFTER_JOURNAL command allows two possible order types:

- ♦ SEQUENTIAL − AIJ records are ordered by physical DBKEY in a AREA:PAGE:LINE sequence. This is the traditional method used by the RMU /OPTIMIZE /AFTER_JOURNAL command and is the default.
- ♦ SCATTER − AIJ records are ordered by a sort key of PAGE:AREA:LINE (page number, area number and line number). This order often allows the RMU /RECOVER command to perform much more effective I/O prefetching and writing to multiple storage areas simultaneously (typically where storage areas of the database are distributed among multiple disk devices).

SCATTER ordering tends to allow more disk devices to be active during the recovery process. This, in turn, should help reduce idle CPU time and allows the recovery to complete in less time. However, because database configurations vary widely, Oracle recommends that you perform tests with both SCATTER and SEQUENTIAL ordering of the optimized after−image journals to determine which method produces the best results for your system.

Note that because an optimized AIJ file is not functionally equivalent to the original AIJ file, the original AIJ file should not be discarded after it has been optimized.

You cannot use optimized AIJ files with the following types of recovery operations:

- ♦ By−area recovery operations (recovery operations that use the RMU Recover command with the Areas qualifier).
- ♦ By−page recovery operations (recovery operations that use the RMU Recover command with the Just_Corrupt qualifier).
- ♦ RMU Recover commands with the Until qualifier. The optimized AIJ does not retain enough of the information from the original AIJ for such an operation.
- ♦ Recovery operation where the database or any storage areas (or both) are inconsistent with the optimized AIJ file. A database or storage area will be inconsistent with the optimized AIJ file if the transaction sequence number (TSN) of the last committed transaction of the database or storage area is not equal to the TSN of the last committed transaction in the open record of the AIJ file. The last committed TSN in the optimized file represents the last transaction committed to the database at the time the original AIJ file was created.

As a workaround for these restrictions against using optimized AIJ files in these recovery operations, use the original, unoptimized AIJ files in these situations instead. Oracle recommends that you do not discard the original AIJ file after it has been optimized.

When using an optimized after−image journal for recovery, the optimal number of buffers specified with the "/AIJ_BUFFERS" qualifier depends on the number of "active" storage areas being recovered. For those journals optimized with "/RECOVERY_METHOD=SEQUENTIAL" (the default), a buffer count of perhaps 250 to 500 is usually sufficient.

When using journals optimized with "/RECOVERY_METHOD=SCATTER", reasonable performance can usually be attained with a buffer count of about five times the number of "active" storage areas being recovered (with a minimum of perhaps 250 to 500 buffers).

"Active" storage areas refers to those areas that have records with modifications reflected in the optimized after−image journal. If only a small number of storage areas are actively modified, less recovery buffers may be needed. The CPU cost of using "excessive" numbers of buffers tends to be relatively small.

When using non−optimized after−image journals for recovery, the RMU /DUMP /AFTER_JOURNAL command can be used to suggest an optimal number of recovery buffers. In effectively all cases though, the default of 20 buffers is probably not sufficient for best performance. Oracle suggests specifying a buffer count of at least 5000 for most databases as a reasonable starting point.

The number of asynchronous prefetch (APF) buffers is also a performance factor during recovery. For recovery operations of optimized after−image journals, the RMU /RECOVER command sets the number of APF buffers (also known as the APF "depth") based on the values of the process quotas ASTLM, BYTLM and the specified "/AIJ_BUFFERS" value. Specifically, the APF depth is set to the maximum of:

- ♦ 50% of the ASTLM process quota
- ♦ 50% of the DIOLM process quota
- ♦ 25% of the specified "/AIJ_BUFFERS" value

Further, accounts and processes that perform RMU /RECOVER operations should be reviewed to ensure that various quotas are set to ensure high levels of I/O performance. Table 9−2 lists suggested quota values for recovery performance.

Table 9−2 Recommended Minimum Process Quotas

Quota	Setting
DIOLM	Equal to or greater than half of the count of database buffers specified by the "/AIJ_BUFFERS" qualifier. Minimum of 250.
BIOLM	Equal to or greater than the setting of DIOLM.
ASTLM	Equal to or greater than 50 more than the setting of DIOLM.
BYTLM	Equal to or greater than 512 times the database buffer size times one half the value of database buffers specified by the "/AIJ_BUFFERS" qualifier. Based on a 12 block buffer size and the desire to have up to 100 asynchronous I/O requests outstanding (either reading or writing), the minimum suggested value is 614,400 for a buffer count of 200.
WSQUOTA, WSEXTENT	Large enough to avoid excessive page faulting
FILLM	50 more than the count of database storage areas and snapshot storage areas.

The RMU /DUMP /AFTER_JOURNAL command indicates the type of optimization (sequential or scattered) when dumping an optimized after−image journal file. The first record in the after−image journal is an "Open" record and contains an indication that the journal is optimized and what type of optimization was specified as shown in the following example (the line reading "Type is Optimized"):

```
1/1   TYPE=O, LENGTH=510, TAD=27−MAY−2003 08:25:31.67, CSM=00
    Database DPA86:[AIJOPT]MF_PERSONNEL.RDB;1
    Database timestamp is 10−DEC−1996 10:17:31.13
    Facility is "RDMSAIJ ", Version is 711.1
    Database version is 71.0
    AIJ Sequence Number is 17231
    Last Commit TSN is 0:1384096
    Synchronization TSN is 0:0
    Journal created on VMS platform
    Type is Optimized (Scatter)
    Open mode is Initial
    Journal was backed up on 27−MAY−2003 08:26:26.25
    Backup type is Quiet−Point
    I/O format is Block
    Commit−to−Journal optimization disabled
    Switchover by process 2023D558
    AIJ journal activation ID is 00A207B1F9111F6B
    LogMiner is enabled
```

9.3.26 Enhancements to INSERT ... FILENAME for LIST OF BYTE VARYING Data

Enhancement 2738471

The INSERT INTO CURSOR ... FILENAME statement loads the contents of the specified file into the LIST OF BYTE VARYING column. In prior versions, the user could specify BINARY or TEXT as the type of data being inserted. This release of Oracle Rdb V7.1 now includes a new type, CHARACTER VARYING.

- BINARY
  Used to load unformatted data such as images, audio files, etc. The contents are broken into 512 octet segments during INSERT.
- TEXT
  Used to load text, a terminator is added to each segment loaded. The contents are written one line to a segment with trailing terminators carriage return (CR) and line feed (LF).
- CHARACTER VARYING
  Used to load text but with no terminator. The contents are written one line to a segment.

In addition, this release allows the TEXT and CHARACTER VARYING source to contain segments of up to 65500 bytes in length. In prior releases, the upper limit was 512 octets.

Interactive SQL now also reports the number of segments inserted and the length of the longest segment. To disable this output, use the SET DISPLAY NO ROW COUNT statement.

The following example shows a sample session that inserts a large text file into a single LIST OF BYTE VARYING column.

```
SQL> create table samples (a list of byte varying);
SQL>
SQL> declare a insert only table cursor for select a from samples;
SQL> declare b insert only list cursor for select a where current of a;
SQL>
SQL> open a;
SQL> insert into cursor a default values;
1 row inserted
SQL>
SQL> open b;
SQL> insert into cursor b
cont>   filename 'WEEKLY_REPORT.DAT' as character varying;
47706 segments inserted (maximum length 270)
SQL> close b;
SQL>
SQL> close a;
```

This statement can only be used in interactive SQL and dynamic SQL.

9.3.27 Index Estimation

Predicate estimation is being used increasingly in the Rdb optimizer to determine the cost and productivity of various index scans.

When a particular query is executed, the conditions in the record select expression, the "where" clause of an SQL statement, determine which rows will be selected. These conditions, or predicates, can be used to limit the parts of an index that are scanned to find data records.

Consider the following SQL query.

```
SQL> SELECT * FROM employees WHERE last_name='Toliver'
cont>  and first_name='Alvin';
```

If there were an index on first_name and a second index on last_name, then Rdb would have to decide which of the two indices would be most efficient for retrieving the data. To do this, Rdb examines each index to find out roughly how many rows would be found through that index. For example, how many 'Toliver' rows would be found in the last_name index.

Historically, Rdb uses estimation in the dynamic optimizer. The dynamic optimizer uses estimation to calculate costs of index scans on competing indices. This information is used to ensure the most efficient indices are scanned first.

During request compilation, the Rdb optimizer uses the selectivity of each expression to help cost various retrieval strategies to determine the most efficient method for retrieving the data.

Where an expression compares a literal value (e.g. WHERE field1=42), and an index exists on that field, the static optimizer can use estimation to obtain from the actual data an estimate for that predicate. In other words, how many rows would actually have the value forty−two in the field called "field1"?

This feature is called ***Sampled Selectivity*** and is described in Section 12.1.1.

9.3.27.1 How Estimation Is Performed

Normally, Rdb performs estimation on indexes of *TYPE IS SORTED* and *TYPE IS SORTED RANKED* by descending the index structure to locate the index node where the selected range spans more than one key value in the node. This is termed the split level.

The *REFINE_ESTIMATES* flag can be used to modify the behaviour of the estimation process. For a complete description of the estimation process, including the new features, please refer to the Rdb Technical Note available through MetaLink.

The *REFINE_ESTIMATES* flag does not change the behaviour of the estimation process for indexes of *TYPE IS SORTED.*

For ranked indexes of *TYPE IS SORTED RANKED*, the *REFINE_ESTIMATES* flag allows estimation to descend beyond the split level to obtain a far more accurate estimate and enables rules to be enforced for how far estimation should proceed.

In addition to the new functionality, the execution trace has been significantly enhanced. In particular, the use of *SET FLAGS 'EXECUTION,DETAIL(1)'* will display significantly more information about the estimation process.

In the following example, the indexes contain the unique values 1 to 100,000. Index I21 is *TYPE IS SORTED* and index I22 is *TYPE IS SORTED RANKED.*

A cursor is opened selecting a range of exactly two keys for two records. In the first open, the keys are such that the estimation descends all the way to the level one node and therefore correctly estimates the number of records from each index as two.

However, in the second open, the keys were chosen such that they happened to span a separator in the index root node. So even though the query would only find two rows in the index, the estimation was erroneously high.

```
SQL> set flags 'strategy,detail(1),exec'
SQL> declare :a,:b,:c,:d int;
SQL> begin
cont> set :a=1; set :b=2; set :c=1; set :d=2;
cont> end;
SQL> declare c1 cursor for select count(*) from t2
cont> where f1 between :a and :b
cont> and f2 between :c and :d;
SQL> open c1;
~S#0003
Tables:
  0 = T2
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:T2 Card=100000
  Bool: (0.F1 >= <var0>) AND (0.F1 <= <var1>) AND (0.F2 >= <var2>) AND (0.F2 <=
        <var3>)
  BgrNdx1 I21 [1:1] Fan=17
    Keys: (0.F1 >= <var0>) AND (0.F1 <= <var1>)
  BgrNdx2 I22 [1:1] Fan=17
    Keys: (0.F2 >= <var2>) AND (0.F2 <= <var3>)
~Estim  Ndx1 Sorted: Split lev=1, Seps=2 Est=2 Precise
~Estim  Ndx2 Ranked: Nodes=1, Min=2, Est=2 Precise IO=3
~Estim  RLEAF Cardinality=  1.0000000E+05
~E#0003.01(1) Estim   Index/Estimate 1/2 2/2
~E#0003.01(1) BgrNdx1 EofData  DBKeys=2  Fetches=0+0  RecsOut=0 #Bufs=1
~E#0003.01(1) BgrNdx2 FtchLim  DBKeys=0  Fetches=0+0  RecsOut=0
~E#0003.01(1) Fin     Buf      DBKeys=2  Fetches=0+1  RecsOut=2
SQL> close c1;
SQL> begin
cont> set :a=35287; set :b=35288; set :c=6207; set :d=6208;
cont> end;
SQL> open c1;
~Estim  Ndx1 Sorted: Split lev=4, Seps=1 Est=5534
~Estim  Ndx2 Ranked: Nodes=309, Min=0, Est=6205 IO=0
~Estim  RLEAF Cardinality=  1.0000000E+05
~E#0003.01(2) Estim   Index/Estimate 1/5534 2/6205
~E#0003.01(2) BgrNdx1 EofData  DBKeys=2  Fetches=1+0  RecsOut=0 #Bufs=1
~E#0003.01(2) BgrNdx2 FtchLim  DBKeys=0  Fetches=0+0  RecsOut=0
~E#0003.01(2) Fin     Buf      DBKeys=2  Fetches=0+1  RecsOut=0
SQL> close c1;
```

The next example shows the difference in the ranked index estimate on I22 when refinement is enabled.

```
SQL> select count(*) from t2
cont> where f1 between :a and :b
cont> and f2 between :c and :d;
~S#0004
Tables:
  0 = T2
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:T2 Card=100000
  Bool: (0.F1 >= <var0>) AND (0.F1 <= <var1>) AND (0.F2 >= <var2>) AND (0.F2 <=
        <var3>)
  BgrNdx1 I21 [1:1] Fan=17
    Keys: (0.F1 >= <var0>) AND (0.F1 <= <var1>)
  BgrNdx2 I22 [1:1] Fan=17
    Keys: (0.F2 >= <var2>) AND (0.F2 <= <var3>)
~Estim  Ndx1 Sorted: Split lev=4, Seps=1 Est=5534
~Estim  Ndx2 Ranked: Nodes=2, Min=2, Est=2 Precise IO=2
~Estim  RLEAF Cardinality=  1.0000000E+05
```

```
~E#0004.01(1) Estim   Index/Estimate 2/2 1/5534
```

Notice that, in this case, the estimate for index I22, which is background index 2, is now precisely two even though the previous query indicated the split level for the same range was very high in the index.

The ranked estimation is more accurate because estimation refinement descended the index beyond the split level until a reasonable estimate was obtained. The estimate for sorted index I21 is still inaccurate because estimation refinement is only available for ranked indices.

Refinement rules allow control of the estimation process by limiting the total IO based on the smallest estimate obtained for each execution of a request. If the first index estimates that 10 rows will be returned, the estimation process can be limited to 10 IO's, on the assumption that fetching the records should take not more than ten IO's.

The ranked index estimation refinement rules are:

- ♦ REFINE_ESTIMATES(1) − Use IO to limit the descend to split level.
- ♦ REFINE_ESTIMATES(2) − Use IO to limit the refinement, where we read beyond the split level.
- ♦ REFINE_ESTIMATES(4) − Limit refinement until the known true branches of an index contain more records than branches of the index that are not completely included in the range selected.
- ♦ REFINE_ESTIMATES(8) − As each node in the index is processed for estimation, the error in that estimate is calculated. This rule terminates estimation once the calculated error in the estimate is less than ten percent of the estimate.
- ♦ REFINE_ESTIMATES(16) − Enable refinement. If this rule is the only refinement rule enabled, refinement will attempt to obtain a precise estimate regardless of the cost. If other refinement rules are also enabled, those rules will be enforced and this rule is not required.

Estimation refinement rules can be combined by adding their values. In this way, all ranked estimation refinement rules can be enabled by using *SET FLAGS 'REFINE_ESTIMATES(15)'*. The *REFINE_ESTIMATES(16)* refinement rule is not required when any of the other rules are enabled.

The following examples show how refinement rules affect estimation on the two ranked indices I22 and I23.

```
SQL> select count(*) from t2 where f2 between 6207 and 6208
cont>    and f3 between 1 and 2;
~S#0003
Tables:
  0 = T2
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:T2 Card=100000
  Bool: (0.F2 >= 6207) AND (0.F2 <= 6208) AND (0.F3 >= 1) AND (0.F3 <= 2)
  BgrNdx1 I22 [1:1] Fan=17
    Keys: (0.F2 >= 6207) AND (0.F2 <= 6208)
  BgrNdx2 I23 [1:1] Fan=17
    Keys: (0.F3 >= 1) AND (0.F3 <= 2)
~Estim  Ndx1 Ranked: Nodes=2, Min=2, Est=2 Precise IO=4
~Estim  RLEAF Cardinality=  1.0000000E+05
~Estim  Ndx2 Ranked: Nodes=0, Min=0, Est=12250 Descend IO limit IO=4
~E#0003.01(1) Estim   Index/Estimate 1/2 2_12250
```

Because the estimate for the first background index is two rows, the estimation process is limited to two IO's. The second background index does not get estimated because estimation on the first index has already used 4 IO's.

```
SQL> select count(*) from t2 where f2 between 6207 and 6208
cont>    and f3 between 1 and 2;
~S#0004
Tables:
  0 = T2
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:T2 Card=100000
  Bool: (0.F2 >= 6207) AND (0.F2 <= 6208) AND (0.F3 >= 1) AND (0.F3 <= 2)
  BgrNdx1 I22 [1:1] Fan=17
    Keys: (0.F2 >= 6207) AND (0.F2 <= 6208)
  BgrNdx2 I23 [1:1] Fan=17
    Keys: (0.F3 >= 1) AND (0.F3 <= 2)
~Estim  Ndx1 Ranked: Nodes=2, Min=2, Est=2 Precise IO=0
~Estim  RLEAF Cardinality=  1.0000000E+05
~Estim  Ndx2 Ranked: Nodes=154, Min=0, Est=3103 Descend IO limit IO=2
~Estim  RLEAF Cardinality=  1.0000000E+05
~E#0004.01(1) Estim   Index/Estimate 1/2 2/3103
```

Because we immediately repeat the same query, the first index is again estimated at two rows, but because the previous query already read these index nodes, this did not cost any IO's because the nodes remained in our buffer pool. This meant that two IO's could be used to begin estimation on the second background index. The estimate is not very accurate because we could only descend a short way into the index structure in two IO's.

In the following example, you will see that we descend further and further down the index on each execution and the estimate becomes progressively better on each execution.

```
SQL> select count(*) from t2 where f2 between 6207 and 6208
cont>    and f3 between 1 and 2;
~S#0005
Tables:
  0 = T2
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:T2 Card=100000
  Bool: (0.F2 >= 6207) AND (0.F2 <= 6208) AND (0.F3 >= 1) AND (0.F3 <= 2)
  BgrNdx1 I22 [1:1] Fan=17
    Keys: (0.F2 >= 6207) AND (0.F2 <= 6208)
  BgrNdx2 I23 [1:1] Fan=17
    Keys: (0.F3 >= 1) AND (0.F3 <= 2)
~Estim  Ndx1 Ranked: Nodes=2, Min=2, Est=2 Precise IO=0
~Estim  RLEAF Cardinality=  1.0000000E+05
~Estim  Ndx2 Ranked: Nodes=1, Min=0, Est=10 Descend IO limit IO=2
~Estim  RLEAF Cardinality=  1.0000000E+05
~E#0005.01(1) Estim   Index/Estimate 1/2 2/10
~E#0005.01(1) BgrNdx1 EofData  DBKeys=2  Fetches=0+0  RecsOut=0 #Bufs=1
~E#0005.01(1) BgrNdx2 FtchLim  DBKeys=0  Fetches=0+0  RecsOut=0
~E#0005.01(1) Fin     Buf      DBKeys=2  Fetches=0+0  RecsOut=0

           0
1 row selected
SQL> select count(*) from t2 where f2 between 6207 and 6208
cont>    and f3 between 1 and 2;
~S#0006
Tables:
  0 = T2
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:T2 Card=100000
  Bool: (0.F2 >= 6207) AND (0.F2 <= 6208) AND (0.F3 >= 1) AND (0.F3 <= 2)
  BgrNdx1 I22 [1:1] Fan=17
    Keys: (0.F2 >= 6207) AND (0.F2 <= 6208)
  BgrNdx2 I23 [1:1] Fan=17
    Keys: (0.F3 >= 1) AND (0.F3 <= 2)
~Estim  Ndx1 Ranked: Nodes=2, Min=2, Est=2 Precise IO=0
~Estim  RLEAF Cardinality=  1.0000000E+05
~Estim  Ndx2 Ranked: Nodes=1, Min=2, Est=2 Precise IO=0
~Estim  RLEAF Cardinality=  1.0000000E+05
~E#0006.01(1) Estim   Index/Estimate 1/2 2/2
```

In this case, after three executions the estimate obtained is precisely two.

It is anticipated that refinement rules will become the default in a future version of Oracle Rdb.

9.3.28 Hash Index Estimation

In addition to the functionality described in Section 9.3.27, the index estimation process has been enhanced to support estimation of indexes of *TYPE IS HASHED*.

For a complete description of this feature, please refer to the Rdb Technical Note available through MetaLink.

By default, Rdb does not allow estimation of hashed indexes. As with the new features for *TYPE IS SORTED RANKED*, this feature is controlled using the refine estimates flag.

The values that effect the behaviour on hashed indexes are:

- REFINE_ESTIMATES(32) − Enable estimation on hashed indexes.
- REFINE_ESTIMATES(64) − Use the smallest estimate obtained for this execution of the request to limit the IO consumed during estimation.

The values can be combined with those affecting ranked indices by adding them together.

In the following query, index I32 is *TYPE IS SORTED RANKED* and index I33 is *TYPE IS HASHED*.

```
SQL> set flags 'strategy,detail(1),exec,refine_estimates(111)'
SQL> select count(*) from t2 where f2 between 6207 and 6208
cont> and f3 in (1,2);
~S#0004
Tables:
  0 = T2
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:T2 Card=100000
  Bool: (0.F2 >= 6207) AND (0.F2 <= 6208) AND ((0.F3 = 1) OR (0.F3 = 2))
  BgrNdx1 I23 [(1:1)2] Fan=1
    Keys: r0: 0.F3 = 2
          r1: 0.F3 = 1
  BgrNdx2 I22 [1:1] Fan=17
    Keys: (0.F2 >= 6207) AND (0.F2 <= 6208)
~Estim  Ndx1 Hashed: Nodes=0, Est=2 Precise IO=5
~Estim  Ndx2 Ranked: Nodes=0, Min=0, Est=12250 Descend IO limit IO=5
~E#0004.01(1) Estim   Index/Estimate 1/2 2_12250
```

The hashed index is background index 1 and is estimated to return precisely two rows. The Refinement rules are in place for ranked indexes so the estimation on the ranked index I22 is not performed.

By repeating the same query, so that some index nodes are already buffered, we can see that estimation on the ranked index will proceed.

Please note that this example has been edited for brevity.

```
SQL> select count(*) from t2 where f2 between 6207 and 6208
cont> and f3 in (1,2);
~Estim  Ndx1 Hashed: Nodes=0, Est=2 Precise IO=0
~Estim  Ndx2 Ranked: Nodes=163, Min=0, Est=3284 Refine IO limit IO=2
~Estim  RLEAF Cardinality=  1.0000000E+05
~E#0005.01(1) Estim   Index/Estimate 1/2 2/3284
~E#0005.01(1) BgrNdx1 EofData  DBKeys=2  Fetches=0+0  RecsOut=0 #Bufs=1
~E#0005.01(1) BgrNdx2 FtchLim  DBKeys=0  Fetches=0+0  RecsOut=0
~E#0005.01(1) Fin     Buf      DBKeys=2  Fetches=0+0  RecsOut=0

           0
1 row selected
SQL> select count(*) from t2 where f2 between 6207 and 6208
cont> and f3 in (1,2);
~Estim  Ndx1 Hashed: Nodes=0, Est=2 Precise IO=0
~Estim  Ndx2 Ranked: Nodes=10, Min=0, Est=190 Refine IO limit IO=2
~Estim  RLEAF Cardinality=  1.0000000E+05
~E#0006.01(1) Estim   Index/Estimate 1/2 2/190
~E#0006.01(1) BgrNdx1 EofData  DBKeys=2  Fetches=0+0  RecsOut=0 #Bufs=1
~E#0006.01(1) BgrNdx2 FtchLim  DBKeys=0  Fetches=0+0  RecsOut=0
~E#0006.01(1) Fin     Buf      DBKeys=2  Fetches=0+0  RecsOut=0

           0
1 row selected
SQL> select count(*) from t2 where f2 between 6207 and 6208
cont> and f3 in (1,2);
~Estim  Ndx1 Hashed: Nodes=0, Est=2 Precise IO=0
~Estim  Ndx2 Ranked: Nodes=2, Min=2, Est=2 Precise IO=0
~Estim  RLEAF Cardinality=  1.0000000E+05
~E#0007.01(1) Estim   Index/Estimate 1/2 2/2
```

Unlike estimation on sorted indexes, estimation of indexes of *TYPE IS HASHED* is performed even where the index has more than one partition.

Estimation on hashed indexes is also supported for range list queries. A range list query, such as the one shown above, provides multiple key values to be retrieved from the same index. This occurs where the query has a condition such as *KEY=42 OR KEY=6* or *KEY IN (1,5,9).*

9.3.29 New LIKE Clause Added to CREATE TABLE

Description

The CREATE ... LIKE statement allows a database administrator to copy the metadata for an existing table and create a new table with similar characteristics.

Syntax

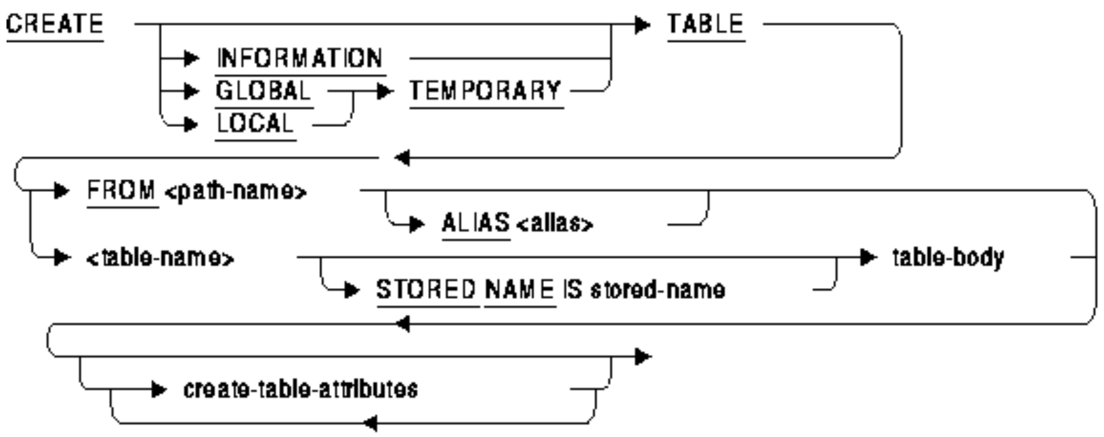
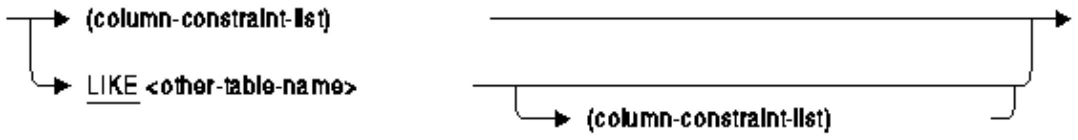


table-body =



Usage Notes

- The table name provided by the LIKE clause must be a base table, a global temporary table, or a local temporary table that currently exists in the current database. Specifying a synonym for a base table or temporary table is also permitted.
  The following attributes of the table are copied:
  - ◊ The names and ordering of all columns
  - ◊ For each column, the data type, DEFAULT, IDENTITY, COMPUTED BY clause, AUTOMATIC AS clause, COMMENT and domain will be inherited.
  - ◊ Display attributes such as DEFAULT VALUE, QUERY NAME, QUERY HEADER and EDIT STRING clauses.
  - ◊ The table comment is inherited, unless overwritten by a COMMENT IS clause.
  - ◊ If the source table includes an IDENTITY column then the LIKE clause will result in a new sequence being created with the same name as this new table.

  Other table attributes such as referential constraints, triggers, storage maps and indices are not inherited and must be separately created.

Note

If a COMPUTED BY expression uses a subselect to reference the current table, then this information is inherited unchanged by the new table. You should perform a subsequent ALTER TABLE statement to DROP and

redefine the COMPUTED BY column.

- ♦ You cannot reference a system table or a view with the LIKE clause.

```
SQL> create table my_sys like rdb$database;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-E-NOMETSYSREL, operation illegal on system defined metadata
```

- ♦ The referenced table name can be followed by a set of additional column names and table constraints. These are added to those inherited from the referenced table. See the example below.

Example: Using the LIKE clause to make a copy of a table definition

This new table will be used to record the EMPLOYEES details after they are retired from the company. An extra column, RETIRED_DATE, is added to record the date of the retirement and a new CHECK constraint is added to ensure that the employee is not listed in both the EMPLOYEES table and this new RETIRED_DATE column.

```
SQL> set dialect 'sql99';
SQL>
SQL> create table RETIRED_EMPLOYEES
cont>     like EMPLOYEES
cont>     (retired_date DATE_DOM
cont>     ,primary key (EMPLOYEE_ID)
cont>     ,check (not exists
cont>       (select * from EMPLOYEES e
cont>        where e.employee_id = RETIRED_EMPLOYEES.employee_id))
cont>      initially deferred
cont>     );
SQL>
SQL> show table RETIRED_EMPLOYEES;
Information for table RETIRED_EMPLOYEES

Columns for table RETIRED_EMPLOYEES:
Column Name         Data Type   Domain
-----------         ---------   ------
EMPLOYEE_ID                     CHAR(5)      ID_DOM
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
RETIRED_DATE                    DATE VMS     DATE_DOM

Table constraints for RETIRED_EMPLOYEES:
RETIRED_EMPLOYEES_CHECK1
 Check constraint
 Table constraint for RETIRED_EMPLOYEES
 Evaluated on COMMIT
 Source:
 CHECK (not exists
    (select * from EMPLOYEES e
      where e.employee_id = RETIRED_EMPLOYEES.employee_id))
```

```
RETIRED_EMPLOYEES_PRIMARY1
 Primary Key constraint
 Table constraint for RETIRED_EMPLOYEES
 Evaluated on UPDATE, NOT DEFERRABLE
 Source:
 PRIMARY key (EMPLOYEE_ID)

Constraints referencing table RETIRED_EMPLOYEES:
No constraints found

Indexes on table RETIRED_EMPLOYEES:
No indexes found

Storage Map for table RETIRED_EMPLOYEES:
No Storage Map found

Triggers on table RETIRED_EMPLOYEES:
No triggers found

SQL>
```

9.3.30 Enhancements to Statistical Functions

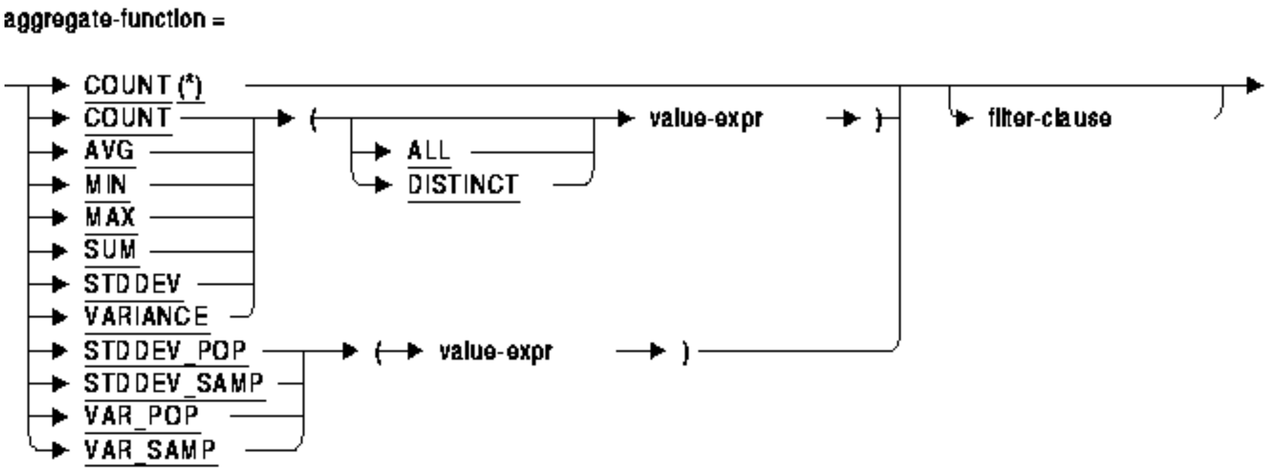
This release of Oracle Rdb V7.1 adds the following new statistical functions to the existing functions COUNT, MAX, MIN, AVG, SUM, STDDEV, and VARIANCE:

- ♦ VAR_POP
  This function calculates the variance for the population. It is equivalent to VARIANCE with degrees of freedom fixed at 0 (for example SET FLAGS 'VARIANCE_DOF(0)' which is the default setting).
- ♦ VAR_SAMP
  This function calculates the variance for a subset or sampling of the population. It is equivalent to VARIANCE with degrees of freedom fixed at 1 (for example SET FLAGS 'VARIANCE_DOF(1)'). By convention, one degree of freedom is used when the sampling of the population is performed.
- ♦ STDDEV_POP
  This function calculates the standard deviation (the square root of the variance) for the population. It is equivalent to STDDEV with degrees of freedom fixed at 0 (for example SET FLAGS 'VARIANCE_DOF(0)' which is the default setting).
- ♦ STDDEV_SAMP
  This function calculates the standard deviation (the square root of the variance) for the subset of sampling of the population. It is equivalent to STDDEV with degrees of freedom fixed at 1 (for example SET FLAGS 'VARIANCE_DOF(1)'). By convention, one degree of freedom is used when the sampling of the population is performed.

These functions are based on the proposed new SQL Database Language standard. The SET FLAGS 'VARIANCE_DOF' option does not change the result of these functions and is only applied to the STDDEV and VARIANCE functions.

In addition, a new FILTER clause is provided for all statistical functions. This clause can be used to limit the values included in the COUNT, MAX, MIN, SUM, AVG, STDDEV and VARIANCE functions.

Syntax



Usage Notes

- ♦ The keywords ALL and DISTINCT are not permitted when using the VAR_POP, VAR_SAMP, STDDEV_POP and STDDEV_SAMP statistical functions.
- ♦ The function COUNT(value−expr) is equivalent to COUNT(*) FILTER (value−exp IS NOT NULL).
  If you apply FILTER to COUNT(value−expr) then it will implicitly include an IS NOT NULL restriction. This might be seen when using the STRATEGY and DETAILS option of SET FLAGS.
- ♦ The FILTER predicate may not include subselect clauses or references to statistical functions.

Examples

The following example shows the differences in results when using these two functions. Use of either function will depend on the application and data being processed. It may be that processing as a sampling might yield a better standard deviation.

Example 9−5 Comparing the output of STDDEV_POP and STDDEV_SAMP

```
SQL> select stddev_pop (salary_amount), stddev_samp (salary_amount)
cont> from SALARY_HISTORY;

  1.777268393871476E+004    1.778488626348816E+004
1 row selected
SQL>
```

FILTER can be used to eliminate data from the statistical function so that the generated report can process the data in a single pass.

Example 9−6 Applying the FILTER clause

```
SQL> select
cont>   max (salary_amount) filter (where salary_end is null),
cont>   max (salary_amount) filter (where salary_end is not null),
cont>   min (distinct salary_amount) filter (where salary_end = salary_start),
cont>   min (distinct salary_amount) filter (where salary_end > salary_start)
cont> from
cont>   salary_history
cont> where
cont>   employee_id = '00164'
cont> group by
cont>   employee_id;

      51712.00        50000.00            NULL        26291.00
1 row selected
SQL>
```

9.3.31 RMU /VERIFY Enhanced to Detect Sequence Problems

RMU /VERIFY has been enhanced to detect two sequence problems. Each sequence that is created has an CLTSEQ entry and a row in the system table RDB$SEQUENCES corresponding to it.

RMU /VERIFY now makes sure that each row in the RDB$SEQUENCES table has a matching CLTSEQ entry in the root, and each CLTSEQ entry in the root file that is marked "Reserved" (i.e. it is associated with a sequence that is being used) has a matching row in the RDB$SEQUENCES table.

On finding these problems, messages of the following type will be generated:

```
%RMU−E−NOSEQROW, Sequence id 1 has an entry in the root file but no row
in RDB$SEQUENCES
%RMU−E−NOSEQENT, Sequence id 1 has no valid entry in the root file
```

This RMU /VERIFY enhancement is available starting with Oracle Rdb Release 7.1.2.

9.3.32 Determining Which Oracle Rdb Options Are Installed

When installing Oracle Rdb Server on OpenVMS you can choose from five components to install:

1. Oracle Rdb
2. Programmer for Rdb (Rdb Compilers)
3. Hot Standby
4. Power Utilities
5. Common Components

Starting with Rdb 7.0, you can determine what Rdb options were selected during the installation of Rdb by running the program SYS$SYSTEM:RDBINS<RdbVersionVariant>.EXE. For example:

```
$ RUN SYS$SYSTEM:RDBINS71
Installed: Oracle Rdb,Rdb Compilers,Hot Standby,Power Utilities
```

Previously, however, the output of the RDBINS program could not easily be redirected. Attempts to redefine SYS$OUTPUT would not allow the program output to be captured.

This problem has been resolved. The RDBINS program now allows redirection of the output to SYS$OUTPUT. The RDBINS program also creates a DCL symbol RDB$INSTALLED_SELECTIONS containing the same output string as is displayed to SYS$OUTPUT.

9.3.33 New Procedure RDB$IMAGE_VERSIONS.COM

The command procedure RDB$IMAGE_VERSIONS.COM is supplied in SYS$LIBRARY by the Rdb installation procedure. The RDB$IMAGE_VERSIONS command procedure can be used to display the image identification string and image link date/time from various Oracle Rdb or potentially related images in SYS$SYSTEM, SYS$LIBRARY and SYS$MESSAGE. This procedure can be used to determine exactly what images are installed on the system.

RDB$IMAGE_VERSIONS.COM accepts an optional parameter. If passed, this parameter specifies a specific file or wildcard to lookup and display information for. By default, filenames starting with RD*, SQL*, RM*, and COSI* and ending with .EXE are searched for and displayed.

The following example shows how to use the RDB$IMAGE_VERSIONS command procedure.

```
Decrdb RTA1:> @RDB$IMAGE_VERSIONS
SYS$SYSROOT:[SYSEXE]RDB$NATCONN71.EXE;1 SQL*NET V7.1−55   8−MAY−2002 15:56
SYS$COMMON:[SYSEXE]RDBINS.EXE;4         ORACLE RDB V7.0  14−NOV−2002 17:20
SYS$COMMON:[SYSEXE]RDBINS70.EXE;33      ORACLE RDB V7.0   7−MAR−2003 15:30
SYS$COMMON:[SYSEXE]RDBINS71.EXE;5       ORACLE RDB V7.1   9−APR−2003 10:58
SYS$COMMON:[SYSEXE]RDBPRE.EXE;5         V7.0−65          10−SEP−2002 16:02
SYS$COMMON:[SYSEXE]RDBPRE70.EXE;37      V7.0−7           28−FEB−2003 23:24
SYS$COMMON:[SYSEXE]RDBPRE71.EXE;5       V7.1−101          8−APR−2003 16:49
SYS$COMMON:[SYSEXE]RDBSERVER.EXE;9      RDB/RSV V7.0−65   5−SEP−2002 21:01
SYS$COMMON:[SYSEXE]RDBSERVER70.EXE;41   RDB/RSV V7.0−7   27−FEB−2003 17:29
SYS$COMMON:[SYSEXE]RDBSERVER71.EXE;5    RDB/RSVV7.1−101   7−APR−2003 17:43
SYS$COMMON:[SYSEXE]RDMABS.EXE;5         RDB V7.0−65      10−SEP−2002 16:01
    .
    .
    .
```

9.3.34 Oracle Rdb SGA API

Oracle Rdb maintains an extensive set of online performance statistics that provide valuable dynamic information regarding the status of an active database. The system global area (SGA) application programming interface (API) described in this document provides a way to retrieve these database performance statistics.

The SGA API automates retrieving database statistics available only through the RMU Show Statistics command. The SGA API provides the only way to retrieve statistics for Oracle Rdb databases from an application. Using the SGA API provides fast access to the data without affecting the execution of the server.

Previously, the Oracle Rdb SGA API was available as a separate software option to be downloaded, installed and maintained independently of the Oracle Rdb kit. Each time a new version of Oracle Rdb was installed, the SGA API would have to be updated. If the SGA API was not updated, it would in many cases fail to work correctly.

This problem has been partly resolved. Most of the contents of the SGA API separate software option are now automatically provided in the RDM$DEMO directory during the Oracle Rdb kit installation procedure. Please refer to the SGA API documentation available in RDM$DEMO in various formats as SGAAPI.PS, SGAAPI.HTML and SGAAPI.TXT for additional information.

Existing Users of the SGAAPI May Have to Modify Procedures

Existing users of the Oracle Rdb SGA API should refer to the documentation as there will be some minor changes required. In particular, the KUSRMUSHRxx.EXE sharable image is now provided in SYS$LIBRARY and the KUSRMUSHRxx.OPT linker options file has been updated to reference this sharable image in its new location.

Chapter 10
Enhancements Provided in Oracle Rdb Release 7.1.1

10.1 Enhancements Provided in Oracle Rdb Release 7.1.1

10.1.1 Scan Intrusion Security Now Supported

Oracle Rdb now supports intrusion detection via the OpenVMS Security Services. See the "OpenVMS Guide to System Security" manual for more information on this feature.

This change is available in Oracle Rdb Release 7.1.1. Customers are encouraged to upgrade to this version to pick up this security enhancement.

Chapter 11
Enhancements Provided in Oracle Rdb Release 7.1.0.4

11.1 Enhancements Provided in Oracle Rdb Release 7.1.0.4

11.1.1 New Keyword SCREEN_NAME for RMU/SHOW STATISTICS/OPTIONS

Bug 2395102

Most RMU Show Statistics screens have a Write option. The use of this option enables the user to capture the current screen to a file named RMU.SCR.

The use of the new keyword Screen_Name option allows you to identify the screen capture by screen name. For example, if you issue an RMU Show Statistics/Option=Screen_Name command, the screen capture is written to a file that has the name of the screen with all spaces, brackets, and slashes replaced by underscores.The file will have a .SCR extension.

For example, if you use the Option=Screen_Name option and select the Write option on the screen Transaction Duration (Read/Write), the screen is written to a file named TRANSACTION_DURATION_READ_WRITE.SCR.

This feature is available in Oracle Rdb Release 7.1.0.4.

11.1.2 Zoom Option for "Process Analysis" Screen in RMU/SHOW STATISTICS

Bug 2395153

A "Zoom" option has been added to the "Process Analysis" screen in RMU/SHOW STATISTICS. The user will now be able to zoom−in on PIDs on this screen. This feature is available in Oracle Rdb Release 7.1.0.4.

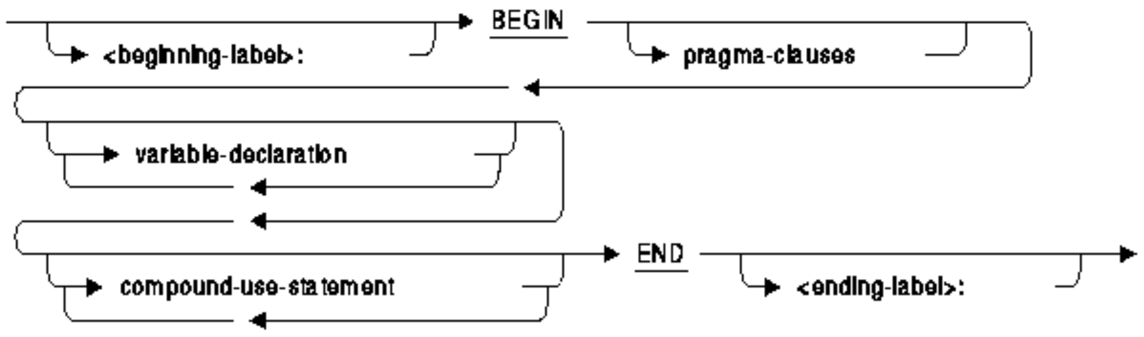
11.1.3 New PRAGMA Clause Added to SQL Compound Statements

A new PRAGMA clause has been added to the compound statement to simplify programming in the SQL Precompiler.

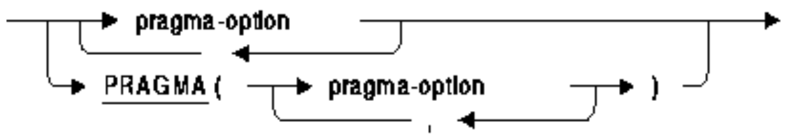
Format

---

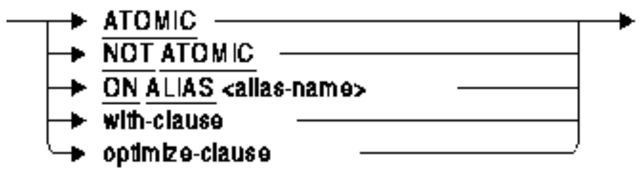
compound-statement =



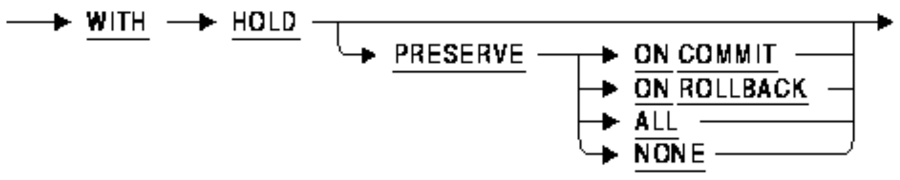
pragma-clauses =



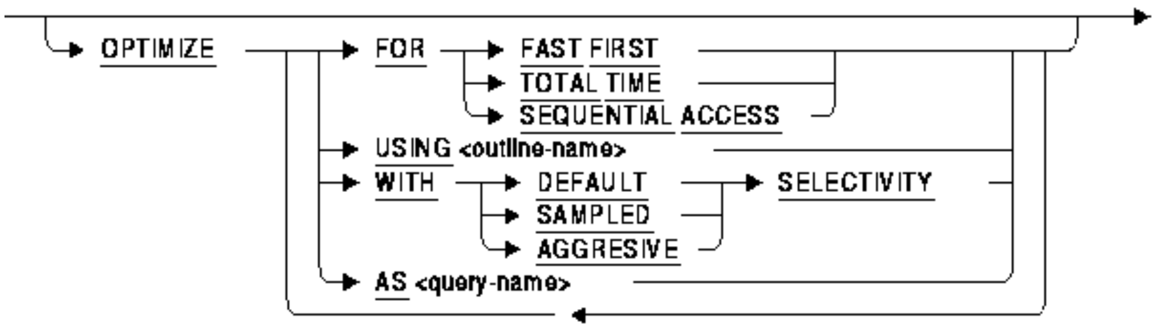
pragma-option =



with-clause =



optimize-clause =



USAGE NOTES

- ♦ The SQL Precompiler also supports the syntax BEGIN DECLARE SECTION. This clause is ambiguous because of the BEGIN DECLARE of the compound statement. Therefore, within the EXEC−SQL compound statement only one pragma clause can be selected. The use of the PRAGMA list allows all options to be specified.
- ♦ The clauses ON ALIAS, OPTIMIZE and WITH HOLD must only appear on the outermost BEGIN of a compound statement.

11.1.4 New DECLARE Routine Statement

Declares a routine as forward reference for database definition statements. A routine is either a function or a procedure.

Description

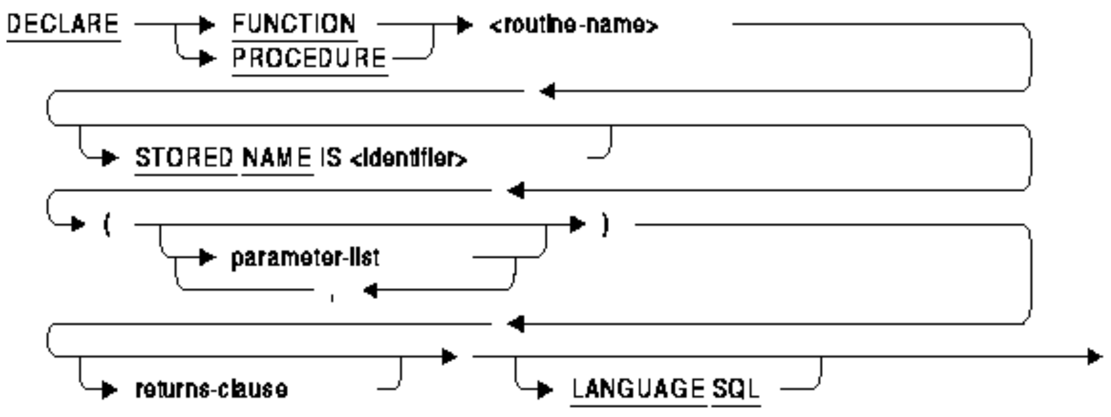
The declared routine acts as a template for calls to the function or procedure in DDL statements such as CREATE TABLE, CREATE VIEW, and CREATE MODULE. The template allows Rdb to validate that the routine is correctly named, is passed the correct number of parameters, and that those parameters are passed compatible arguments. For functions, the returned data type is used to calculate data types for COMPUTED BY, AUTOMATIC, and other stored value expressions.

Environment

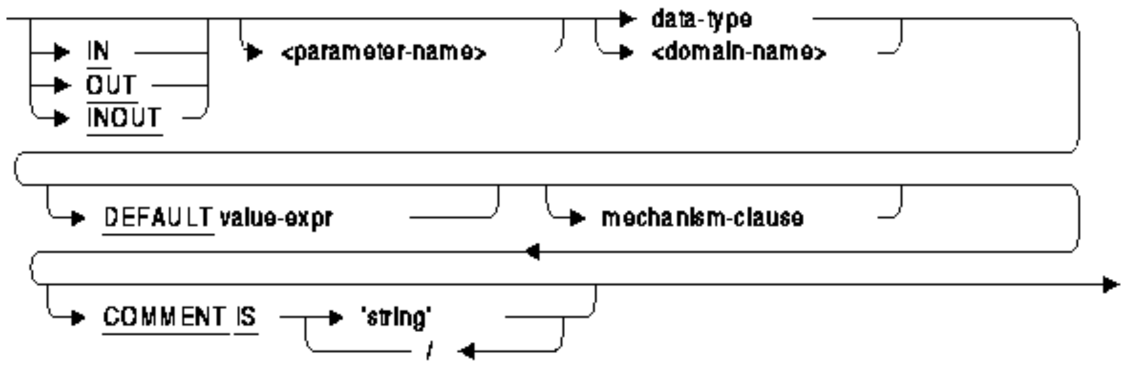
You can use the DECLARE FUNCTION and DECLARE PROCEDURE statements:

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

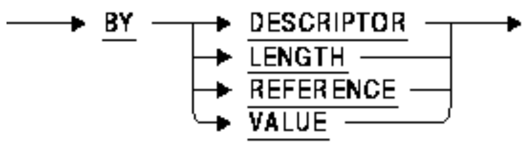
Format



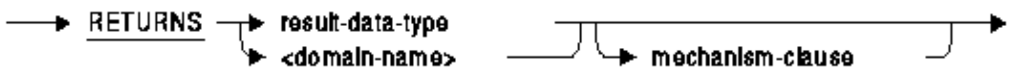
parameter-list =



mechanism-clause =



returns-clause =



Usage Notes

- If an additional DECLARE statement is executed with the same routine name, then it must be identical to the existing definition.
- The routine that is created using CREATE FUNCTION, CREATE PROCEDURE, or CREATE MODULE statements must match exactly the number of parameters, the data types (domains can be replaced with the base data types), passing mechanism (BY VALUE, BY REFERENCE, BY LENGTH, BY DESCRIPTOR), and mode (IN, OUT and INOUT).
- The DEFAULT clause on parameters must be specified so that the minimum and maximum parameter counts can be calculated for the routine. However, this DEFAULT value is not used and may be specified as NULL, i.e. a placeholder.
- A declared routine remains part of the session until it is replaced by a CREATE FUNCTION, CREATE PROCEDURE, or CREATE MODULE statement.
  If a CREATE FUNCTION, CREATE PROCEDURE, or CREATE MODULE statement is rolled back, then any declared routine it replaced is also eliminated. Therefore, a new DECLARE will be required in such cases.
- If the session is disconnected before a CREATE statement has defined the true routine body (stored or external), then attempts to use the database objects which reference those routines will fail.
  This is similar to the behavior observed after using DROP ... CASCADE. i.e. there are unresolved references which must be corrected by creating those objects.
- Tools such as SQL EXPORT and IMPORT and RMU Extract use the DECLARE routine facility to allow forward references in generated database definition operations.
  For RMU Extract, the Item=Forward_References qualifier must be used to enable the output of the DECLARE statements. For SQL EXPORT, this is the default setting which can be disabled using the NO FORWARD_REFERENCES clause with the EXPORT or IMPORT commands.

Example

Consider this simple definition of a domain and referencing external function.

```
SQL> create domain MONEY as integer (2);
SQL>
SQL> create function INTEREST_PAID
cont>    (in :amt MONEY)
cont>    returns MONEY;
cont>    external
cont>       language C
cont>       parameter style GENERAL;
SQL>
SQL> alter domain MONEY
cont>    add
cont>       check (INTEREST_PAID (value) > 0)
cont>       not deferrable;
```

Once the ALTER DOMAIN is completed, neither the function nor the domain can be defined before the other. Here is an excerpt of the result of executing the output from the RMU Extract command.

```
SQL> create domain MONEY
cont>     INTEGER (2)
cont>     check((INTEREST_PAID(value) > 0))
cont>     not deferrable;
%SQL−F−RTNNOTDEF, function or procedure INTEREST_PAID is not defined
SQL>
SQL> commit work;
SQL> create function INTEREST_PAID (
cont>     in    :AMT
cont>         MONEY
cont>         by reference)
cont>     returns
cont>         MONEY by value
cont>     language SQL;
cont>     external
cont>         language C
cont>         parameter style GENERAL
cont>     deterministic
cont>     called on null input
cont>     ;
%SQL−F−NO_SUCH_FIELD, Domain MONEY does not exist in this database or schema
SQL> commit work;
```

This problem is avoided for RMU Extract by adding the Forward_References item to the command line:

```
$ RMU/EXTRACT/ITEM=(ALL,FORWARD_REFERENCES) databasename/OUTPUT=script.SQL
```

The script now contains a forward declaration of the function INTEREST_PAID so that execution of the script can succeed.

```
SQL> declare function INTEREST_PAID (
cont>     in    :AMT
cont>         INTEGER (2)
cont>         by reference)
cont>     returns
cont>         INTEGER (2) by value
cont>     ;
SQL>
SQL> create domain MONEY
cont>     INTEGER (2)
cont>     check((INTEREST_PAID(value) > 0))
cont>     not deferrable;
SQL>
SQL> commit work;
SQL> create function INTEREST_PAID (
cont>     in    :AMT
cont>         MONEY
cont>         by reference)
cont>     returns
cont>         MONEY by value
cont>     language SQL;
cont>     external
cont>         language C
cont>         parameter style GENERAL
cont>     deterministic
cont>     called on null input
cont>     ;
SQL> commit work;
```

11.1.5 New AUTO_INDEX Option Added for SET FLAGS

This release of Oracle Rdb includes a new AUTO_INDEX option for the SET FLAGS statement and

RDMS$SET_FLAGS logical. This option can be used to have CREATE TABLE and ALTER TABLE create indices for any PRIMARY KEY, FOREIGN KEY or UNIQUE constraint added to the table.

Note

This feature is part of a prototyping facility and is not intended to replace database design and management. Many example scripts, such as the PetStore demonstration for JDBC, assume that adding constraints will also implicitly create indices for performance. In such examples, simply include SET FLAGS 'AUTO_INDEX' in the script that creates the database.

The following example shows actions of AUTO_INDEX:

```
SQL> set dialect 'SQL92';
SQL> set flags 'AUTO_INDEX,INDEX_STATS';
SQL> create table PERSON
cont> (employee_id      integer primary key,
cont>  manager_id       integer references PERSON (employee_id),
cont>  last_name        char(30),
cont>  first_name       char(30),
cont>  unique (last_name, first_name));
~Ai create index "PERSON_PRIMARY_EMPLOYEE_ID"
~Ai larea length is 430
~Ai storage area (default) larea=57
~Ai create sorted index, ikey_len=5
Sort    Get     Retrieval sequentially of relation PERSON
~Ai create index partition, node=430 %fill=0
~Ai create index "PERSON_FOREIGN1"
~Ai larea length is 215
~Ai storage area is shared: larea=57
~Ai create sorted index, ikey_len=5
Sort    Get     Retrieval sequentially of relation PERSON
~Ai create index partition, node=0 %fill=0
~Ai create index "PERSON_UNIQUE1"
~Ai larea length is 215
~Ai storage area is shared: larea=57
~Ai create sorted index, ikey_len=62
Sort    Get     Retrieval sequentially of relation PERSON
~Ai create index partition, node=0 %fill=0
SQL>
SQL> show table (index) person
Information for table PERSON

Indexes on table PERSON:
PERSON_FOREIGN1                 with column MANAGER_ID
  Duplicates are allowed
  Type is Sorted
  Key suffix compression is DISABLED

PERSON_PRIMARY_EMPLOYEE_ID      with column EMPLOYEE_ID
  No Duplicates allowed
  Type is Sorted
  Key suffix compression is DISABLED
  Node size  430

PERSON_UNIQUE1                  with column LAST_NAME
                                and column FIRST_NAME
  Duplicates are allowed
  Type is Sorted
  Key suffix compression is DISABLED
SQL>
```

Usage Notes

- ♦ All indices which are created for constraints are of type SORTED. If the database SYSTEM INDEX default is SORTED RANKED then this same default is used by the AUTO_INDEX option.
- ♦ If a suitable index already exists, then it will be used in preference to creating a new index.
- ♦ All indices are created in the DEFAULT storage area. There is no facility to add storage maps for these indices during their creation.
- ♦ The index is given the same name as the constraint for which it was created. When the constraint is dropped, the index will remain and must be dropped manually. It is possible that the index is used by multiple constraints.
- ♦ Use the INDEX_STATS option with AUTO_INDEX to see a description of the indices which are created.

Chapter 12
Improving Query Performance Using Sampled Selectivity

12.1 Sampled Selectivity

12.1.1 Improving Query Performance Using Sampled Selectivity

Oracle Rdb Release 7.1.2 introduces a new optimizer feature called ***sampled selectivity***. Use of sampled selectivity can improve the performance of certain database queries by giving the Rdb optimizer more accurate information about the distribution of data in tables. The role of the Rdb optimizer is to decide how to execute a query on an Rdb database in the most efficient manner. To accomplish that, the optimizer considers different ways of joining results from various tables and different methods of retrieving the data for each table. The result of the optimization process is a ***query strategy***. The choice of query strategy depends primarily on its cost, based on estimating the number of disk I/O operations that will be done when the query is executed. This might be mitigated by constraints placed on the query, such as, fast first execution and query outlines.

12.1.2 Selectivity in the Optimization Process

Most of the time the Rdb optimizer does an excellent job of choosing query strategies. However, query optimization is not an exact science; and in a small number of cases the chosen strategies are sub−optimal (the queries take longer to perform than one might expect). Sometimes the reason for this can be traced to dramatically inaccurate estimates of the number of data rows that will be processed. Sampled rather than static computation of selectivity might help correct this.

Part of computing the I/O cost involves breaking down a SQL query's WHERE clause into individual predicates. For instance, A.LAST_NAME = B.LAST_NAME is a predicate that specifies a join condition on two tables. Another example is the predicate EMPLOYEE_ID = '00164', where a column value is restricted to a range, or in this case, to a single value. The purpose of such predicates is to limit the number of rows to be retrieved. Each predicate specifies some condition that allows only those rows that qualify to be selected. The ratio of the number of qualifying rows divided by the total number of rows is called the ***selectivity factor***.

The Rdb optimizer determines predicate selectivity in different ways. For example, given a join condition, such as, A.LAST_NAME = B.LAST_NAME, the selectivity is computed as some function of the cardinality of the two tables. For predicates of the form EMPLOYEE_ID = '00164' (more particularly "column−name relational−operator literal−value" or "column−name IS NULL"), the optimizer assigns a fixed value to the selectivity, as described in the next section. It is for this second type of expression, when an index exists on the column, that sampled selectivity can be employed.

12.1.2.1 Fixed Selectivity Is Used by Default

For predicates of the form "column−name relational−operator literal−value" or "column−name IS NULL", the optimizer assigns a selectivity factor whose value depends on the operator used in the expression. In the example EMPLOYEE_ID = '00164', the default behavior of Rdb is to assign the equals operator a fixed selectivity factor of 3.125%. If the EMPLOYEES table has 100 rows in it, the equals operation is assumed on average to return three rows (which is 3%). This selectivity value is fixed and does not depend on the value '00164' in the predicate. If instead the predicate were EMPLOYEE_ID = '00273', the optimizer would still predict that three rows in the EMPLOYEES table would match that selection criterion.

Rdb provides two choices for fixed selectivity values; the first is the default set for Rdb and the other choice uses ***aggressive selectivity*** values that predict fewer rows will be returned than do the standard Rdb values. Given an estimated cost for retrieving all the data in a selected table column, selectivity is used to reduce that cost to some fraction representing the subset of data that is of interest.

12.1.2.2 Fixed Selectivity Can Sometimes Be a Poor Predictor

Selectivity is used to predict cardinality (the number of rows to be processed), and predicted cardinality is used to estimate I/O cost. However, fixed selectivity can be a poor predictor given certain distributions of data. For example, if a table of 100 rows has a COUNTRY column and all values in the table for that column happen to be 'SWITZERLAND', the true selectivity of the predicate WHERE COUNTRY = 'SWITZERLAND' should be 100%, not 3%. This is a case where the distribution of column values over the range of possible values is very narrow. Such a data distribution is said to be highly skewed.

12.1.3 Introducing Sampled Selectivity

Oracle Rdb Release 7.1.2 introduces another way to calculate selectivity. This can be done by sampling the data in a table's index and estimating the cardinality from the actual distribution of data. Given an estimate of a predicate's cardinality and given the number of rows in a table, one can estimate the selectivity factor. Tests have shown that, on average, selectivity estimation done by sampling data in an index is more accurate than by simply using a fixed value.

Execution of the Rdb optimizer is divided into a static optimization phase and a dynamic optimization phase. The role of the static optimizer is to choose the "best" query strategy. For certain types of queries the dynamic optimizer processes several competing indices. When multiple indexes on a table exist, the dynamic optimizer accesses those which are useful to see which is the most productive. Different executions of the query can thus adapt to changing input parameters and use the best index.

The first step in dynamic optimization involves sorting background indexes in order by the number of expected returned keys. The number of index entries to be processed is estimated by sampling each candidate index. The static optimizer now uses that same method of sampling an index to estimate the cardinality of a result and from that to compute the selectivity.

12.1.3.1 Pros and Cons of the Different Selectivity Methods

Sampled selectivity computation is usually more accurate than using a fixed selectivity value because it is based on the actual distribution of data in a table. A sorted, ranked index gives better overall results than does a sorted, non−ranked index because cardinality information is stored within a sorted, ranked index. For any given data value, any one of the three methods (fixed, sorted, ranked) might yield the most accurate result. Typically, ranked indexes give the best results and fixed selectivity gives the least accurate results.

The following is a simple range query on the EMPLOYEES table in the sample PERSONNEL database.

```
SQL> select employee_id from employees where employee_id > 'nnnnn';
```

There are 100 rows in the EMPLOYEES table. The values in the EMPLOYEE_ID column are unique and range from '00164' to '00471'. The worst estimates for all three methods occur when 'nnnnn' = '00164'. The fixed selectivity method predicts 35 rows, so it is too low by 65 rows. The sorted, unranked index method predicts 34 rows, so it is too low by 66 rows. The ranked index method predicts 84 rows, but it is only wrong by 16 rows.

Sampled selectivity comes with a small cost. In order to get the more accurate estimates, the optimizer must sample indexes during query compilation; and this might require I/O operations to be performed depending on how many of the index nodes are already in memory. If the indexes used for the estimation are also used during query execution, there might be no additional I/O if the same index nodes must be referenced again as the index information will typically remain buffered. Also, if the query is compiled once but executed many times, any additional I/O to perform the estimation might be insignificant.

There is no evidence to show that enabling aggressive selectivity for all queries will result in overall better query strategies than by using the standard (default), fixed Rdb values. Aggressive selectivity is best used for specific queries where it is shown to help and where sampled selectivity cannot be used.

12.1.3.2 Requirements for Using Sampled Selectivity

When the feature is enabled, sampled selectivity estimation is attempted only for certain forms of predicate and only under the right set of circumstances. For example, if a table has no indexes, selectivity cannot be estimated by sampling since there are no indexes on which to sample the data. The following is a set of rules that define when sampled selectivity estimation can and cannot be performed.

- Predicate Form
  The predicate must be one of the following types:
  column = literal
  column <> literal
  column > literal
  column >= literal
  column < literal
  column <= literal
  column IS NULL
  For all but the IS NULL case, the operands can be transposed, for example, literal = column. When predicates use variables instead of literals, estimation by sampling cannot be done because the value is unknown at query compilation time.
- Base Table Column
  The column reference must be to a base table column, such as the EMPLOYEE_ID column in the EMPLOYEES table, or to a view column that maps one−to−one with such a column.
- Column Is First Index Segment
  At least one sorted or sorted, ranked index on the table must exist with the predicate's column as its first segment. The index may have more than one segment.
- Hashed Index not Used
  For an index to be useful in the estimation process, it must be either a sorted index or a sorted, ranked index. Hashed indexes are not used for sampled selectivity estimation.
- Single Partition Index
  For an index to be considered useful, it can only have a single partition.
- Ascending Key Values

In the candidate indexes, all columns must be sorted in ascending order.

- No Explicit Collating Sequence
  The column used for the first index segment must not have any explicit collating sequence specified.
- No Mapping Values
  The first index segment must not be mapped (see the MAPPING VALUES clause in the SQL CREATE INDEX statement).

In the future it might be possible to relax or eliminate some of the preceding restrictions given sufficient interest in so doing.

12.1.4 How to Enable the Various Selectivity Methods

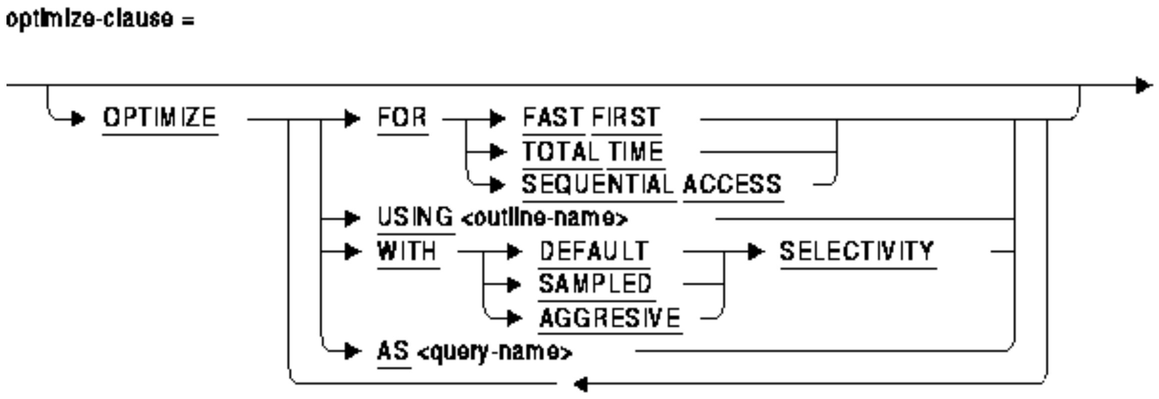
Rdb uses several different methods for estimating predicate selectivity. These methods are shown in Table 12−2. Two of those methods, (Fixed) and (Index), can now be influenced by you. If you do nothing, by default Rdb will use it's standard set of values for assigning (Fixed) selectivity to predicates. There are several ways to specify the method to be used:

- For individual queries
  By including the optional OPTIMIZE WITH clause on INSERT ... SELECT, SELECT, UPDATE, DELETE, and compound statements (only on the outermost BEGIN−END block), you can specify the type of selectivity computation to use for individual queries. See Section 12.1.4.1.
- For queries made within the current SQL session
  You can avoid having to include the OPTIMIZE WITH clause on each SQL query by establishing a default method for selectivity calculation. For interactive and dynamic SQL see Section 12.1.4.2, which describes the SET OPTIMIZATION LEVEL statement and Section 12.1.4.3, which describes the SET FLAGS statement. For pre−compiled SQL and for SQL module language code see Section 12.1.4.4 to read about compiler switches that affect selectivity estimation and Section 12.1.4.3 for a description of the SET FLAGS statement.
- For all query sessions
  See Section 12.1.4.3 for the use of the RDMS$SET_FLAGS logical name.
- For RMU/UNLOAD
  A qualifier has been added to the RMU/UNLOAD command that allows you to specify how selectivity is to be evaluated. See Section 12.1.4.5.

12.1.4.1 OPTIMIZE WITH Clause

The INSERT ... SELECT, SELECT, UPDATE, DELETE and compound statements (only at the outer BEGIN−END block) have an optional clause, OPTIMIZE WITH, that specifies what type of selectivity computation method is to be used. The OPTIMIZE FOR, OPTIMIZE USING, and OPTIMIZE AS forms of the OPTIMIZE clause are already described in the Oracle Rdb SQL Reference Manual.

Format



When using the OPTIMIZE WITH clause, you can specify one of three options. If you choose sampled selectivity, the Rdb optimizer will use the index sampling method for selectivity estimation wherever possible. For those predicates where this is not possible, the norm is to use standard, fixed selectivity values. If you choose aggressive selectivity, the Rdb optimizer will use the fixed, aggressive values for selectivity computation. Finally, if you choose to use default selectivity for the query, this specifically means that index sampling and aggressive selectivity will not be used.

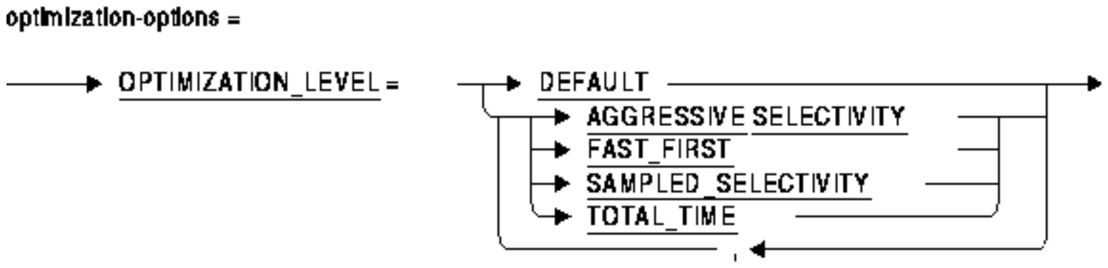
The following example shows how to use this new clause.

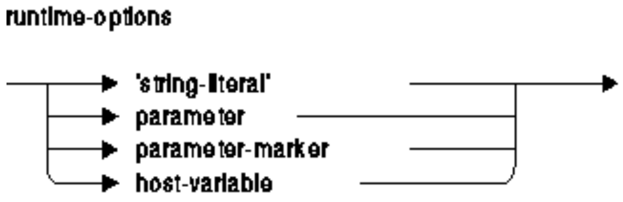
```
SQL> select * from employees where employee_id < '00170'
cont>   optimize with sampled selectivity;
```

12.1.4.2 SET OPTIMIZATION LEVEL Statement

New options have been added to the SET OPTIMIZATION LEVEL statement, options that allow you to specify default selectivity behavior for your INSERT ... SELECT, SELECT, UPDATE, DELETE and compound statements within the session. The FAST FIRST, TOTAL TIME and DEFAULT options of the SET OPTIMIZATION LEVEL statement are described in the Oracle Rdb SQL Reference Manual.

Format





If you choose other than the DEFAULT behavior, you may pick one of either FAST FIRST or TOTAL TIME and/or you may choose one of the selectivity options. For selectivity computation the chosen default is applied to each INSERT ... SELECT, SELECT, UPDATE, DELETE, and compound statement in the SQL session provided that those statements do not explicitly have the OPTIMIZE WITH clause, described in Section 12.1.4.1.

If you set optimization level for sampled selectivity, the Rdb optimizer will use the index sampling method for selectivity estimation wherever possible. For those predicates where this is not possible, Rdb will revert to using standard, fixed selectivity values. If you set optimization level for aggressive selectivity, the Rdb optimizer will use the fixed, aggressive values for selectivity computation.

12.1.4.3 The SELECTIVITY Debug Flag

Yet another way that you can declare how selectivity is to be computed is by setting the SELECTIVITY debug flag. You can do so in one of two ways. One is by defining the OpenVMS logical name, RDMS$SET_FLAGS. The other is by using the SQL SET FLAGS statement. The RDMS$SET_FLAGS logical name and the SET FLAGS statement are described in the Oracle Rdb SQL Reference Manual.

When you define selectivity using the RDMS$SET_FLAGS logical name, it affects queries for all SQL sessions which are run within the scope of that logical name. Doing so can be useful when trying to debug query performance problems. When you define selectivity using the SET FLAGS statement, its effect lasts for the duration of the database attach.

The SELECTIVITY debug flag lets you specify default, aggressive, or sampled selectivity behavior, just as you can using the OPTIMIZE WITH clause. In addition, the SELECTIVITY flag lets you enable both aggressive and sampled behavior, something that is not allowed with the SET OPTIMIZATION LEVEL statement nor the OPTIMIZE WITH clause. The SELECTIVITY debug flag affects queries that do not otherwise have a selectivity mode specified. It can also affect partial query outlines, triggers, constraints, and internal Rdb queries.

The SELECTIVITY debug flag takes a numeric argument, with a value of from 0 to 3. For example,

```
$ DEFINE RDMS$SET_FLAGS "SELECTIVITY(2)"
```

or

```
SQL> SET FLAGS 'SELECTIVITY (2)';
```

Table 12−1 shows the numeric values for the SELECTIVITY debug flag and their meanings.

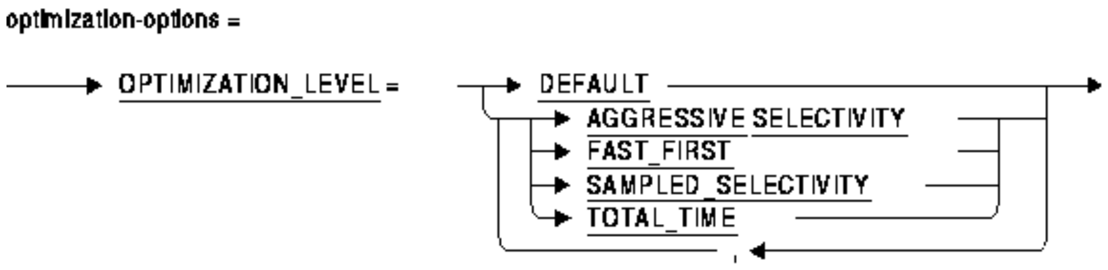
Table 12−1 Settings for the SELECTIVITY Debug Flag

Selectivity Debug Flag Setting	Meaning
SELECTIVITY (0)	Default selectivity
SELECTIVITY (1)	Aggressive selectivity
SELECTIVITY (2)	Sampled selectivity
SELECTIVITY (3)	Sampled + aggressive selectivity

12.1.4.4 SQL Precompiled and SQL Module Language Code

Optimizer selectivity controls can also be enabled using the OPTIMIZATION_LEVEL qualifier on SQL precompiled or SQL Module Language compiled code. In addition to being able to establish default values for TOTAL TIME versus FAST FIRST optimization, the OPTIMIZATION_LEVEL qualifier can now indicate the type of selectivity estimation to perform by default. SELECT, INSERT ... SELECT, UPDATE, DELETE and compound statements (on the outermost BEGIN ... END) will inherit these settings during compilation of the module.

Format



12.1.4.5 RMU/UNLOAD/OPTIMIZE

A SELECTIVITY option has been added to the RMU/UNLOAD/OPTIMIZE command. By specifying the SELECTIVITY option you can direct how the optimizer will estimate predicate selectivity values during RMU/UNLOAD operations.

Syntax:

```
RMU/UNLOAD/OPTIMIZE=(SELECTIVITY:selectivity-option)
```

```
selectivity_option = {DEFAULT, SAMPLED, or AGGRESSIVE}
```

If you choose sampled selectivity, the Rdb optimizer will use the index sampling method for selectivity estimation wherever possible. For those predicates where this is not possible, the norm is to use standard, fixed selectivity values. If you choose aggressive selectivity, the Rdb optimizer will use the fixed, aggressive values for selectivity computation. Finally, if you choose to use default selectivity, this specifically means that index sampling and aggressive selectivity will not be used.

12.1.5 Improving the Accuracy of Sampled Selectivity

Sampled selectivity estimation using a sorted, ranked index can be done with varying degrees of accuracy. More accuracy might require that more I/O be spent during the index sampling process depending on how many of the index nodes are resident in memory. Sampling accuracy is controlled by the REFINE_ESTIMATES debug flag. For more information about refining estimates on sorted, ranked indexes, see Section 9.3.27.

When sampled selectivity is being computed with refined estimates in effect, there is no specific numeric limit placed on the number of I/O operations to be performed. That being the case, there are three refinement rules (as defined in Section 9.3.27) that can affect the calculation of sampled selectivity:

- Rule 3−−−Limit refinement to the TRUE/MIXED cardinality case.
  To enable this refinement rule, SET FLAGS 'REFINE_ESTIMATES(4)' or DEFINE RDMS$SET_FLAGS "REFINE_ESTIMATES(4)".
- Rule 4−−−Limit refinement so that the error in the estimate is within ten percent.
  To enable this refinement rule, SET FLAGS 'REFINE_ESTIMATES(8)' or DEFINE RDMS$SET_FLAGS "REFINE_ESTIMATES(8)".
- Rule 5−−−Try to provide precise estimates.
  To enable this refinement rule, SET FLAGS 'REFINE_ESTIMATES(16)' or DEFINE RDMS$SET_FLAGS "REFINE_ESTIMATES(16)".

Rules 3 and 4 can be combined by setting the value of the REFINE_ESTIMATES flag to 12. Rule 5 is only needed if you want no other limits on the refinement process and want the most precise estimates.

12.1.6 Details about the Sampled Selectivity Process

This section gives details about the operation of the Rdb optimizer as it performs sampled selectivity computation.

In an early stage of query compilation the static optimizer scans each WHERE clause in the query, locates each ***leaf predicate***, and assigns it a selectivity factor. Consider the following: WHERE LAST_NAME = 'Toliver' AND FIRST_NAME = 'Alvin'. Also, assume there exist sorted indexes on the LAST_NAME and FIRST_NAME columns. The leaf predicates are (1) LAST_NAME = 'Toliver', and (2) FIRST_NAME = 'Alvin'. There is a higher−level predicate, the AND of two expressions, which is not a leaf predicate. In this example, sampled selectivity estimation is only performed for the two leaf predicates. The key steps in this process are:

1. Validate the predicate format
   The form of the predicate must be one of those described in Section 12.1.3.2.
2. Avoid Redundant Estimation
   To avoid unnecessary I/O, the optimizer maintains a list of up to 100 predicates for which estimation by sampling has already been done. The list only applies to the current query. If two predicates are the same and are for the same table, the second predicate is given the already−computed selectivity value of the first one.
3. Verify the Presence of One or More Useful Indexes
   The table in which the predicate column exists must have one or more indexes which could be used for sampled selectivity estimation. What determines that an index is useful is explained in Section 12.1.3.2.
4. Choose the Best Index for the Job
   A table can have multiple indexes, and several indexes might be valid for doing the estimation. The optimizer looks at each such index and chooses one using the following criteria.
    1. Ranked over Non−Ranked
       First, a sorted, ranked index is assumed to give more accurate estimates than an index that is not.
    2. Unique over Duplicates Allowed
       If both indexes are equal thus far, and if one index is unique and the other index allows duplicates, the unique index is chosen as likely to give the more accurate results.
    3. Shorter index key Length
       All other things being equal, an index with a shorter index key length is chosen.
5. Estimate Cardinality by Sampling the Index
   Once the optimizer chooses an index it scans the index to estimate the expected cardinality for the predicate. Cardinality is then used to compute selectivity.
6. When Sampled Estimation Fails
   If sampled selectivity estimation is attempted and fails for whatever reason, the optimizer reverts to using the fixed selectivity values.

12.1.7 Diagnostic Information about Selectivity Estimation

If you are interested in seeing details about selectivity estimation there is an option to do so. For each query that is compiled you can see what selectivity values were used, what methods were chosen to derive those values, and cardinality predicted for each predicate (where appropriate). Another option will show you the following: either (1) the name of the index used to estimate selectivity, or (2) the reason(s) that selectivity could not be estimated by sampling.

12.1.7.1 How to Enable Diagnostic Output

Normally, diagnostic information is not shown. To see information about the optimizer's query estimation process, you must enable the ESTIMATES flag. That capability has been available in Rdb for many years. To view details about predicate selectivity computation, you must also enable the DETAIL flag, as described in the following paragraphs. To set these flags, either define the RDMS$SET_FLAGS logical name or use the SET FLAGS statement in SQL.

Normal query estimation summary:

To enable standard output about the query estimation process, define the RDMS$SET_FLAGS logical name as follows:

```
$ DEFINE RDMS$SET_FLAGS ESTIMATES
```

or

```
SQL> SET FLAGS 'ESTIMATES';
```

Examples of the output you might see for this and other flag settings are shown in Section 12.1.7.4.

Query estimation summary plus predicate selectivity:

To enable detailed output about the query estimation process, define the RDMS$SET_FLAGS logical name as follows:

```
$ DEFINE RDMS$SET_FLAGS "ESTIMATES,DETAIL(2)"
```

or

```
SQL> SET FLAGS 'ESTIMATES,DETAIL(2)';
```

With these settings you will see for each query the standard summary about query estimation plus, for each predicate, the selectivity values used, what methods were chosen to derive those values, and cardinality predicted (where appropriate).

Query estimation summary plus predicate selectivity and more:

By using DETAIL(3) in place of DETAIL(2) in the preceding statements, you will also see either (1) the name of the index used to estimate selectivity, or (2) the reason(s) that selectivity could not be estimated by sampling.

12.1.7.2 How to Disable Diagnostic Output

To disable output about the query estimation process, you can either use the RDMS$SET_FLAGS logical name or use the SET FLAGS statement in SQL.

By de−assigning the RDMS$SET_FLAGS logical name, you undo any previously set flags. Among other things, this disables the output of query estimates.

```
$ DEASSIGN RDMS$SET_FLAGS
```

If you want to continue using sampled selectivity but no longer need to see query estimates, do the following instead:

```
$ DEFINE RDMS$SET_FLAGS "SELECTIVITY(2)"
```

When redefining the RDMS$SET_FLAGS logical name, you have to include those flags you still want to remain in effect. By contrast, in interactive or dynamic SQL you do not have to refer to flags you want to remain in effect.

```
SQL> SET FLAGS 'NOESTIMATES,NODETAIL';
```

The preceding disables query estimates but leaves other flags unchanged.

12.1.7.3 Details about Selectivity Estimation Diagnostics

The method for selectivity computation appears in parentheses on the output line showing estimated selectivity (see examples in Section 12.1.7.4). Table 12−2 shows the various methods:

Table 12−2 Methods of Computing Selectivity

Method	Meaning
(Average)	Selectivity was computed using some average statistical value known about the table or an index, e.g., an index segment group factor.
(Cardinality)	Selectivity was computed as some function of current table cardinality.
(Fixed)	A fixed value for selectivity was chosen (either standard or aggressive).
(Index)	Selectivity was computed by sampling an index.
(Index) (Dup)	Selectivity computation was avoided. The predicate is a duplicate of one elsewhere in the query, one for which selectivity was computed by sampling an index.

When selectivity cannot be determined by index sampling, and when the level of detail in the diagnostic output is properly set (see Section 12.1.7.1), messages can appear in the output to explain the reason(s). These messages and what they mean are listed in Table 12−3.

Table 12−3 Reasons That Selectivity Sampling Cannot Be Done

Message	Meaning
Column has an explicit collating sequence	This is explained in Section 12.1.3.2.
Sampled selectivity is disabled	This is self−explanatory.
Error during index scan	It was not possible to perform the index scan. For example, under unusual circumstances a buffer overflow can occur.
Exception error	An unexpected error occurred. This indicates that a design or implementation problem exists in the Rdb optimizer. The problem should be reported to Oracle.
Expression not supported	See Section 12.1.3.2 to see which expressions are acceptable and which

for sampled selectivity	are not.
Indexes not valid for sampled selectivity	For selectivity to be computed by sampling, the predicate column's table must have an index with appropriate attributes, as explained in Section 12.1.3.2.
Internal error ...	An unexpected error occurred. This indicates that a design or implementation problem exists in the Rdb optimizer. The problem should be reported to Oracle.
Operator not supported for sampled selectivity	See Section 12.1.3.2 to see which operators are acceptable and which are not.
Selectivity is fixed for outer join Booleans	The selectivity of an outer join Boolean predicate is fixed at 1.0 (i.e., 100%).
Table has no indexes	For selectivity to be computed by sampling, the predicate column's table must have an index with appropriate attributes, as explained in Section 12.1.3.2.

12.1.7.4 Examples of Selectivity Estimation Diagnostics

This section shows examples of what you might see when you ask for query estimation diagnostics.

Example 1: Normal diagnostics for the query estimation process

The following example shows a simple query and the normal summary one can see for the query estimation process. This occurs when the level of detail is not specified (equivalent to detail level 0). The output of the estimates summary is described in the Oracle Rdb Guide to Database Performance and Tuning, Section C.4, Displaying Optimization Statistics with the O Flag.

```
SQL> set flags 'estimates';
SQL> select last_name from employees where employee_id > '00400';
Solutions tried 2
Solutions blocks created 1
Created solutions pruned 0
Cost of the chosen solution   3.3090658E+00
Cardinality of chosen solution   3.5000000E+01
```

Example 2: Detail level 2 estimates

There is no difference between detail levels 0 and 1 for the estimation diagnostics. At detail level 2, prior to the normal estimates, Rdb displays information about the predicates in the query, in this case, the predicate EMPLOYEE_ID > '00400'. First, each table is listed along with a correlation number. Below, the EMPLOYEES table is given as table 0. Next, each predicate is shown (this query has only one). 0.EMPLOYEE_ID signifies the EMPLOYEE_ID column in table 0, i.e., EMPLOYEES. Following the listed predicate is a line showing the predicate selectivity, the method by which that selectivity value was derived (Index, in this case), and the estimated cardinality for the predicate. Estimated cardinality and selectivity are related by the formula:

```
selectivity = estimated cardinality / table cardinality
```

The notation (Index) means that predicate selectivity was calculated by sampling an index (sampled selectivity).

```
SQL> set flags 'estimates,detail(2)';
SQL> select last_name from employees where employee_id > '00400'
       optimize with sampled selectivity;
~Predicate selectivity and cardinality estimation:
Tables:
  0 = EMPLOYEES
Predicates:
  0.EMPLOYEE_ID > '00400'
    Selectivity  5.9999999E-02 (Index)        Estimated cardinality 6
Solutions tried 2
Solutions blocks created 1
Created solutions pruned 0
Cost of the chosen solution   1.9074016E+00
Cardinality of chosen solution   6.0000000E+00
```

Example 3: Detail level 3 estimates

At detail level 3, the output includes additional information: either (1) the name of the index used to estimate selectivity, or (2) the reason(s) that selectivity could not be estimated by sampling. In this example there are two predicates. For the first predicate, selectivity is computed by index sampling and the name of the index used is shown. For the second predicate, selectivity cannot be computed by sampling because there is no useful index for doing so. Although there is an index on the EMPLOYEE_ID column, there is none on the LAST_NAME column.

```
SQL> set flags 'estimates,detail(3)';
SQL> select last_name from employees
cont> where employee_id < '00180' and last_name > 'W'
      optimize with sampled selectivity;
~Predicate selectivity and cardinality estimation:
Tables:
  0 = EMPLOYEES
Predicates:
  0.EMPLOYEE_ID < '00180'
    Selectivity  1.6000000E-01 (Index)        Estimated cardinality 16
      Index EMP_EMPLOYEE_ID used
  0.LAST_NAME > 'W'
    Selectivity  3.4999999E-01 (Fixed)        Estimated cardinality 35
      Indexes not valid for sampled selectivity
        Index EMP_EMPLOYEE_ID is not useful
          First index segment is not predicate column
Solutions tried 2
Solutions blocks created 1
Created solutions pruned 0
Cost of the chosen solution   2.4074016E+00
Cardinality of chosen solution   5.5999999E+00
```

Example 4: Example showing the (Average) Method for Computing Selectivity

For a predicate of the form column = value, selectivity is determined by sampling if possible, that is, the (Index) method. If this cannot be done and if the table has a single segment index on that column, the method used is (Average). That is, an average value is used for the entire index.

```
SQL> set flags 'estimates,detail(3)';
SQL> select employee_id, last_name from employees
cont> where employee_id = '00250';
~Predicate selectivity and cardinality estimation:
Tables:
  0 = EMPLOYEES
```

```
Predicates:
  0.EMPLOYEE_ID = '00250'
    Selectivity  9.9999998E-03 (Average)      Estimated cardinality 1
      Sampled selectivity is disabled
Solutions tried 2
Solutions blocks created 1
Created solutions pruned 0
Cost of the chosen solution   1.6474016E+00
Cardinality of chosen solution   1.0000000E+00
```

Example 5: Example showing the (Cardinality) Method for Computing Selectivity

For some predicates, the selectivity is a function of table cardinality.

```
SQL> select e.employee_id, e.last_name, d.department_name
cont> from employees e, departments d
cont> where e.employee_id = d.manager_id;
~Predicate selectivity and cardinality estimation:
Tables:
  0 = EMPLOYEES
  1 = DEPARTMENTS
Predicates:
  0.EMPLOYEE_ID = 1.MANAGER_ID
    Selectivity  9.9999998E-03 (Cardinality)
      Expression not supported for sampled selectivity
Solutions tried 10
Solutions blocks created 4
Created solutions pruned 1
Cost of the chosen solution   4.5832439E+01
Cardinality of chosen solution   2.6000000E+01
```

| Contents