Oracle® Rdb for OpenVMS

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Oracle® Rdb for OpenVMS

Release Notes

Release 7.1.0.3

July 2002

Oracle Rdb Release Notes, Release 7.1.0.3 for OpenVMS

Contents

Preface

Purpose of This Manual

This manual contains release notes for Oracle Rdb Release 7.1.0.3. The notes describe changed and enhanced features; upgrade and compatibility information; new and existing software problems and restrictions; and software and documentation corrections.

Intended Audience

This manual is intended for use by all Oracle Rdb users. Read this manual before you install, upgrade, or use Oracle Rdb Release 7.1.0.3.

Document Structure

This manual consists of seven chapters:

Chapter 1	Describes how to install Oracle Rdb Release 7.1.0.3.
Chapter 2	Describes software errors corrected in Oracle Rdb Release 7.1.0.3.
Chapter 3	Describes software errors corrected in Oracle Rdb Release 7.1.0.2.
Chapter 4	Describes software errors corrected in Oracle Rdb Release 7.1.0.1.
Chapter 5	Describes enhancements introduced in Oracle Rdb Release 7.1.0.3.
Chapter 6	Provides information not currently available in the Oracle Rdb documentation set.
Chapter 7	Describes problems, restrictions, and workarounds known to exist in Oracle Rdb Release 7.1.0.3.

Chapter 1
Installing Oracle Rdb Release 7.1.0.3

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

NOTE

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

1.1 Requirements

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

- Oracle Rdb must be shutdown before you install this update kit. That is, the command file SYS$STARTUP:RMONSTOP71.COM should be executed before proceeding with this installation. If you have an OpenVMS cluster, you must shutdown the Rdb 7.1 monitor on all nodes in the cluster before proceeding.
- The installation requires approximately 240,000 blocks for OpenVMS Alpha systems.

1.2 Invoking VMSINSTAL

To start the installation procedure, invoke the VMSINSTAL command procedure:

```
@SYS$UPDATE:VMSINSTAL RDBAMVE3071 device−name OPTIONS N
```

device−name

Use the name of the device on which the media is mounted.

- If the device is a disk drive, such as a CD−ROM reader, you also need to specify a directory. For CD−ROM distribution, the directory name is the same as the variant name. For example:

  ```
  DKA400:[RDBAMVE3071.KIT]
  ```
- If the device is a magnetic tape drive, you need to specify only the device name. For example:

  ```
  MTA0:
  ```

OPTIONS N

This parameter prints the release notes.

The following example shows how to start the installation on device MTA0: and print the release notes:

```
$ @SYS$UPDATE:VMSINSTAL RDBAMVE3071 MTA0: OPTIONS N
```

The full Oracle Rdb Release 7.1.0 Installation Guide is also available on MetaLink in Adobe Acrobat PDF format:

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

1.3 Stopping the Installation

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

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

1.4 After Installing Oracle Rdb

This update provides a new Oracle Rdb Oracle TRACE facility definition. Any Oracle TRACE selections that reference Oracle Rdb will need to be redefined to reflect the new facility version number for the updated Oracle Rdb facility definition, "RDBVMSV7.1−03".

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

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

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

```
$ LIBRARY /TEXT /EXTRACT=RDBVMSV7.1−03 −
_$ /OUT=RDBVMS.EPC$DEF SYS$SHARE:EPC$FACILITY.TLB
```

2. Insert the facility definition into the Oracle TRACE administration database.

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

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

1.5 Alpha EV68 Processor Support Added

For this release of Rdb, Oracle Rdb Release 7.1.0.3, the Alpha EV68 processor is the newest processor supported.

1.6 Maximum OpenVMS Version Check Added

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

OpenVMS Version 7.3−x is the maximum supported version of OpenVMS.

As of Oracle Rdb Release 7.1, the Alpha EV68 processor is supported.

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

Chapter 2
Software Errors Fixed in Oracle Rdb Release 7.1.0.3

This chapter describes software errors that are fixed by Oracle Rdb Release 7.1.0.3.

2.1 Software Errors Fixed That Apply to All Interfaces

2.1.1 Query With Same Column in Two Clauses Returns Wrong Results

Bug 2285818

The following query with the same column in two clauses should return 1 row:

```
set flags 'strategy,detail';

SELECT T2.vert, T3.flag, T5.data
        FROM T1, T2, T3, T4, T5
        WHERE T1.plan_id = T2.plan_id
           AND T1.cust_id = T4.cust_id
           AND T3.prod_id = T4.prod_id
           AND T5.prod_id = T4.prod_id
           AND T3.prod_id = T5.prod_id
           AND T1.code = ' '
           AND ((T4.prio = 3 AND T3.flag = '10')   <== "T3.flag = '10'" is
             OR (T4.prio = 3 AND T3.flag = '12')
             OR (T4.prio = 3 AND T3.flag = '13'))
           AND ((T3.flag = '10' AND T5.data = '73' ) <== reused here again
                 OR T5.data <> '73' ) ;
Tables:
  0 = T1
  1 = T2
  2 = T3
  3 = T4
  4 = T5
Cross block of 5 entries
  Cross block entry 1
    Conjunct: 0.CODE = ' '
    Get     Retrieval sequentially of relation 0:T1
  Cross block entry 2
    Get     Retrieval by index of relation 1:T2
      Index name  I_T2_01 [1:1]  Direct lookup
        Keys: 0.PLAN_ID = 1.PLAN_ID
  Cross block entry 3
    Conjunct: 3.PRIO = 3
    Get     Retrieval by index of relation 3:T4
      Index name  I_T4_01 [1:1]         Direct lookup
        Keys: 0.CUST_ID = 3.CUST_ID
  Cross block entry 4
    Conjunct: (2.FLAG = '10') OR (2.FLAG = '12') OR
              (2.FLAG = '13')
    Get     Retrieval by index of relation 2:T3
      Index name  I_T3_01 [1:1]        Direct lookup
        Keys: 2.PROD_ID = 3.PROD_ID
  Cross block entry 5
    Conjunct: (2.PROD_ID = 4.PROD_ID) AND
              ((4.DATA = '73') OR (4.DATA <> '73')) <== missing "FLAG = '10'"
    Get     Retrieval by index of relation 4:T5
      Index name  I_T5_01 [1:1]        Direct lookup
        Keys: 4.PROD_ID = 3.PROD_ID
 T2.VERT       T3.FLAG  T5.DATA
```

```
 LV_508        13       73                    <== WRONG
 LH_610        12       75                    <== CORRECT
2 rows selected
```

One of the equality predicates in the OR clauses referencing table T3 is referenced again in another clause, as seen below.

```
          AND ((T4.prio = 3 AND T3.flag = '10')   <== "T3.flag = '10'" is
            OR (T4.prio = 3 AND T3.flag = '12')
            OR (T4.prio = 3 AND T3.flag = '13'))
          AND ((T3.flag = '10' AND T5.data = '73' ) <== reused here again
                OR T5.data <> '73' ) ;
```

However, in the detailed strategy display, the predicate is missing under the cross block entry 5, as seen below.

```
  Cross block entry 5
    Conjunct: (2.PROD_ID = 4.PROD_ID) AND
              ((4.DATA = '73') OR (4.DATA <> '73')) <== missing "FLAG = '10'"
    Get     Retrieval by index of relation 4:T5
      Index name  I_T5_01 [1:1]        Direct lookup
        Keys: 4.PROD_ID = 3.PROD_ID
```

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

1. The main query joins 5 tables (T1, T2, T3, T4, T5) using cross strategy with 5 cross block entries. Tables T3, T4 and T5 are joined by PROD_ID column key, table T4 is joined with T1 by CUST_ID, and T1 is joined with T2 by PLAN_ID.
2. Table T1 rows are filtered by an equality predicate. If this is removed, the strategy changes in the order of the cross blocks and the query works.
3. One of the filter predicates contains OR expressions which reference one column of table T4 (T4.PRIO) and one column of table T3 (T3.FLAG).
4. Another filter predicate contains an OR expression which references the same column of table T3 from the previous filter predicate (e.g. T3.FLAG = '10'). This is the main reason why the query returns wrong results.

As a workaround, the query works if the second predicate "T3.FLAG = '10'" is replaced by a LIKE operator, for example "T3.FLAG like '10'".

```
set flags 'strategy,detail';

SELECT T2.vert, T3.flag, T5.data
        FROM T1, T2, T3, T4, T5
        WHERE T1.plan_id = T2.plan_id
           AND T1.cust_id = T4.cust_id
           AND T3.prod_id = T4.prod_id
           AND T5.prod_id = T4.prod_id
           AND T3.prod_id = T5.prod_id
           AND T1.code = ' '
           AND ((T4.prio = 3 AND T3.flag = '10')   <== "T3.flag = '10'" is
             OR (T4.prio = 3 AND T3.flag = '12')
             OR (T4.prio = 3 AND T3.flag = '13'))
           AND ((T3.flag LIKE '10' AND T5.data = '73' ) <== replaced by LIKE
                 OR T5.data <> '73' ) ;
```

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.1.2 GROUP BY Query Followed by CASE With EXISTS Clause Returns Wrong Results

Bug 2198990

The following GROUP BY query followed by CASE with EXISTS clause should return 3 rows but returns only 2 rows.

```
set flags 'strategy,detail';

select count(*), RD.York_Loss_Code,
    CASE WHEN EXISTS (Select * from Loss_Gruppe where
                          Loss_Code = RD.York_Loss_Code
                         )
         THEN 'P'
         ELSE 'F' END
    from redraw RD
group by RD.York_Loss_Code,
    CASE WHEN EXISTS (Select * from Loss_Gruppe where
                          Loss_Code = RD.York_Loss_Code
                         )
         THEN 'P'
         ELSE 'F' END
optimize using test_outline
;
~S: Outline "TEST_OUTLINE" used
Aggregate       Sort
Match
  Outer loop
    Sort    Get     Retrieval sequentially of relation REDRAW
  Inner loop
    Aggregate       Sort    Get
    Retrieval sequentially of relation LOSS_GRUPPE
                YORK_LOSS_CODE
            1   1               P
            1   2               P
2 rows selected
```

The tables contain the following rows:

```
sel york_loss_code from redraw;
 YORK_LOSS_CODE
 1
 10
 2
3 rows selected

sel loss_code from loss_gruppe;
 LOSS_CODE
 1
 2
2 rows selected
```

This feature was not included in the very first release of Oracle Rdb7 and this is the first time the customer has used a GROUP BY clause followed by a CASE with EXISTS clause.

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

1. The main select query is a count aggregate with GROUP BY clause.
2. One of columns in the GROUP BY clause contains a CASE expression with an EXISTS clause on a subquery.

As a workaround, the query works if the query outline TEST_OUTLINE is changed to use cross strategy, as seen below.

```
create outline TEST_OUTLINE
id '1B91E858006B77EC167036406D2D04AB'
mode 0
as (
  query (
    subquery (
      subquery (
        REDRAW 0        access path sequential
          join by cross to
!         join by match to
        subquery (
          LOSS_GRUPPE 1   access path sequential
          )
        )
      )
    )
  )
compliance optional ;
```

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.1.3 ORDER BY Query on a BIGINT or INT Column Returns Wrong Order

Bug 2261391

The following ORDER BY query on a BIGINT column returns values in the wrong order.

```
create data file foo;
create tab t1 (a smallint,b date vms,c bigint);
insert into t1 value (1,  '16-APR-2002 14:13:41.33',  -8214388935822950413);
insert into t1 value (1,  '16-APR-2002 18:54:02.53',   3170710922826741446);
create index i1 on t1 (a,b,c);

select * from t1 order by c;
      A   B                                          C
      1   16-APR-2002 18:54:02.53    3170710922826741446
      1   16-APR-2002 14:13:41.33   -8214388935822950413
2 rows selected
```

A similar error might occur on INT columns and that problem has also been corrected.

As a workaround, disable the QSORT feature by defining the logical RDMS$BIND_MAX_QSORT_COUNT to zero.

```
$DEFINE RDMS$BIND_MAX_QSORT_COUNT 0
```

```
SQL$
SQL>attach 'f foo';
SQL>sel * from t1 order by c;
      A   B                                           C
      1   16-APR-2002 14:13:41.33   -8214388935822950413
      1   16-APR-2002 18:54:02.53    3170710922826741446
2 rows selected
```

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.1.4 OR Clause With Constant Predicate Returns Wrong Results

Bug 2405927

The following query with an OR predicate containing a constant predicate should return a non−zero count.

```
set flags 'strategy,detail';

SELECT COUNT(T2.NR_ITEM)
    FROM T1, T2
    WHERE T2.CD_VENDOR = T1.CD_VENDOR
      AND T2.NR_PROD = T1.NR_PROD
      AND T2.IN_CANCEL = 'N'
      AND T1.IN_PRIO  = 'S'
      AND (T1.CD_VENDOR = '187102' OR '187102' = '' )
      AND (T1.SHIP_DT BETWEEN '1-may-2002 00:00:00'
            AND '6-may-2002 00:00:00' ) ;
Tables:
  0 = T1
  1 = T2
Aggregate: 0:COUNT (1.NR_ITEM)
             Bool: NOT MISSING (1.NR_ITEM)
Cross block of 2 entries
  Cross block entry 1
    Conjunct: 0.IN_PRIO = 'S'
    Conjunct: (0.CD_VENDOR = '187102') OR ('187102' = '')
    Conjunct: 0.SHIP_DT >= '1-MAY-2002'
    Conjunct: 0.SHIP_DT <= '6-MAY-2002'
    Index only retrieval of relation 0:T1
      Index name  T1.NR_DOC_SRT [0:0]
  Cross block entry 2
    Leaf#01 BgrOnly 1:T2 Card=4386
      Bool: (1.CD_VENDOR = 0.CD_VENDOR) AND (1.NR_PROD = 0.NR_PROD) AND
             (1.IN_CANCEL = 'N')
             AND ('187102' = '')        ! Note: <= missing left side of OR
      BgrNdx1 T2.CD_VENDOR_HSH [2:2] Fan=1
        Keys: (1.CD_VENDOR = 0.CD_VENDOR) AND (1.NR_PROD = 0.NR_PROD)
      BgrNdx2 T2.DT_T1.SRT [2:2] Fan=13
        Keys: (1.CD_VENDOR = 0.CD_VENDOR) AND (1.NR_PROD = 0.NR_PROD)
        Bool: 1.IN_CANCEL = 'N'
           0
1 row selected
```

Notice that one of the descendants of the OR predicate is missing in the detail dump of the dynamic leaf strategy.

This is a regression caused by the fix made for Bug 2285818 where a query with shared OR predicate returns wrong results.

As a workaround, the query works if the dynamic strategy is disabled by setting the SQL flag 'MAX_STABILITY' or defining the logical RDMS$MAX_STABILITY as Y.

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.1.5 SELECT COUNT(*) Might Bugcheck Under Certain Dialects of SQL

Bug 2415860

When the dialect was set to SQL92, SQL99 or ORACLE LEVEL1, it was possible for SELECT COUNT(*) to bugcheck when the optimizer used a SORTED RANKED index.

The following example shows the problem.

```
SQL> SET DIALECT 'ORACLE LEVEL1';
SQL> select count(*) from fea_person;
%RDMS-I-BUGCHKDMP, generating bugcheck dump file SYSMAN:[MANAGER]RDSBUGCHK.DMP;
%RDMS-I-BUGCHKDMP, generating bugcheck dump file SYSMAN:[MANAGER]SQLBUGCHK.DMP;
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
virtual address=0000000000000010, PC=0000000000363F1C, PS=0000001B
```

The dump file shows this exception:

```
***** Exception at 00F2FF08 : RDMS$$GEN_SORT_KEY_ASNS + 000015F8
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
virtual address=0000000000000010, PC=0000000000F2FF08, PS=0000000B
```

A workaround for the problem is to disable the count scan optimization used for SORTED RANKED indices:

```
SQL> SET FLAGS 'NOCOUNT_SCAN'
```

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.1.6 Getting Null Values Instead of Actual Values

Bug 2245379

When columns of a table have been added or dropped several times it is possible, in some rare conditions, to get null values instead of actual ones for a column when doing a sequential scan of the table.

The following example shows the different results depending on the strategy.

```
SQL> select f1,f2 from t where f1 > 0 and f1 < 3;
Index only retrieval of relation T
  Index name  I_T [1:1]
          F1            F2
           1             1
           2             1
```

```
SQL> select f1,f2,f3 from t where f1 > 0 and f1 < 3;
Conjunct        Get     Retrieval sequentially of relation T
          F1            F2            F3
           1          NULL             1
           2          NULL             1
```

As a workaround for the problem, add a new column to the table showing the problem.

```
SQL> Alter table t add column xx integer;
SQL> commit;
```

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.1.7 Another OR With Two Constant Predicates Returns Wrong Results

Bugs 2451862, 2405927, 2285818

The following query with OR predicate containing constant predicates should find some rows.

```
set flags 'strategy,detail';

SELECT T2_ANO, T2_ORGAO
   FROM T1, T2 WHERE
      T2_ORGAO  = T1_ORGAO AND
      T2_ANO    = T1_ANO AND
      ((T1_DATA = '187310105' AND T1_ANO  = '02') OR
       ('187310105'  = ' ' AND '02' =  ' ')) AND
      (T2_STATUS  = 'PE');
Tables:
  0 = T1
  1 = T2
Cross block of 2 entries
  Cross block entry 1
    Leaf#01 FFirst 0:T1 Card=1
      Bool: ((0.T1_DATA = '187310105') AND (0.T1_ANO = '02')) OR
             (('187310105' = ' ') AND ('02' = ' '))
      BgrNdx1 NECE_SG_SETOR_IN_SRT [0:0] Fan=18
  Cross block entry 2
    Leaf#02 FFirst 1:T2 Card=2
      Bool: (1.T2_ORGAO = 0.T1_ORGAO) AND
             (1.T2_ANO = 0.T1_ANO) AND
             ('187310105' = ' ') AND ('02' = ' ') <== WRONG
             AND ((1.T2_STATUS = 'PE') OR (1.T2_STATUS <> 'EN'))
      BgrNdx1 T2_SRT [2:2] Fan=11
        Keys: (1.T2_ORGAO = 0.T1_ORGAO) AND (1.T2_ANO =
              0.T1_ANO)
0 rows selected
```

The query works if one of the constant equality predicates is removed, as in the following example.

```
SELECT T2_ANO, T2_ORGAO
    FROM T1, T2 WHERE
      T2_ORGAO  = T1_ORGAO AND
      T2_ANO    = T1_ANO AND
```

```
      ((T1_DATA = '187310105' AND T1_ANO  = '02') OR
!      ('187310105'  = ' ' AND '02' =  ' ')) AND
       ('187310105'  = ' ')) AND
      (T2_STATUS  = 'PE');
Tables:
  0 = T1
  1 = T2
Cross block of 2 entries
  Cross block entry 1
    Leaf#01 FFirst 0:T1 Card=1
      Bool: ((0.T1_DATA = '187310105') AND (0.T1_ANO = '02')) OR
             ('187310105' = ' ')
      BgrNdx1 NECE_SG_SETOR_IN_SRT [0:0] Fan=18
  Cross block entry 2
    Leaf#02 FFirst 1:T2 Card=2
      Bool: (1.T2_ORGAO = 0.T1_ORGAO) AND (1.T2_ANO =
            0.T1_ANO) AND (1.T2_STATUS = 'PE')
      BgrNdx1 T2_SRT [2:2] Fan=11
        Keys: (1.T2_ORGAO = 0.T1_ORGAO) AND (1.T2_ANO =
              0.T1_ANO)
 T2_ANO   T2_ORGAO
 02       187
1 row selected
```

A second update kit to Oracle Rdb 7.0.6.3 included Bug 2405927 where the constant predicate in an OR tree is pulled out of the OR predicate and re−generated in other leg. However, this fix did not cover the current query where there are more than one simple constant predicates in the OR clause, as below.

```
       ('187310105'  = ' ' AND '02' =  ' ')) AND
```

As a workaround, the query works if the dynamic strategy is disabled by setting the SQL flag 'MAX_STABILITY' or defining the logical RDMS$MAX_STABILITY to Y.

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.1.8 Another Query With Same Column in Two Clauses Returns Wrong Results

Bugs 2453935, 2285818

The following query with the same column in two clauses should return 1 row.

```
set flags 'strategy,detail';

SELECT S.PROD_ID,
       S.CONTRACT_ID,
       S.LONG_QTY,
       S.SHORT_QTY
  FROM  SALE S,
        PRODUCT P
  WHERE
        (S.LONG_QTY > 0 OR S.SHORT_QTY > 0) AND
        (S.PROD_ID = P.PROD_ID AND
         (S.SHORT_QTY >  0 OR P.PROD_CODE = 'FUT')) ;
Tables:
  0 = SALE
```

```
  1 = PRODUCT
Cross block of 2 entries
  Cross block entry 1
    Get     Retrieval sequentially of relation 1:PRODUCT
  Cross block entry 2
    Conjunct: ((0.LONG_QTY > 0) OR (0.SHORT_QTY > 0)) AND ((
              0.SHORT_QTY > 0) OR (1.PROD_CODE = 'FUT'))
    Get     Retrieval by index of relation 0:SALE
      Index name  SALE_NDX [1:1]
        Keys: 0.PROD_ID = 1.PROD_ID
 S.PROD_ID   S.CONTRACT_ID   S.LONG_QTY   S.SHORT_QTY
        15           14200   0            0
        15           14207   16           0
2 rows selected
```

This is similar to the query reported in Bug 2285818, where one of the equality predicates in the OR clauses referencing table SALE is referenced again in another clause, as in the following example.

```
        (S.LONG_QTY > 0 OR S.SHORT_QTY > 0) AND    <== "S.SHORT_QTY > 0" is
        (S.PROD_ID = P.PROD_ID AND
         (S.SHORT_QTY >  0 OR P.PROD_CODE = 'FUT')) <== reused here again
```

As a workaround, the query works if the shared predicate is placed at the first position within the OR clause, as in the following example.

```
        (S.SHORT_QTY > 0 OR S.LONG_QTY > 0) AND  <== "S.SHORT_QTY > 0" is 1st
        (S.PROD_ID = P.PROD_ID AND
         (S.SHORT_QTY >  0 OR P.PROD_CODE = 'FUT')) <== and reused here again
```

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.2 SQL Errors Fixed

2.2.1 Unexpected TRANSACTION Debug Output for Compound Statements

In Oracle Rdb Release 7.1.0.2, the output from the TRANSACTION debug flag is always displayed for SET TRANSACTION, START TRANSACTION and LOCK TABLE statements within a compound (BEGIN END) statement. This will occur whenever the compound statement is compiled by the Rdb Server.

This problem can interfere with application execution and, for servers such as SQL*Net for Rdb, can fill output log files.

The following example shows that the TRANSACTION dump is output for LOCK TABLE even though the TRANSACTION flag is not enabled.

```
SQL> begin
cont> lock table employees for shared read mode;
cont> end;
~T Compile transaction (2) on db: 1
~T Transaction Parameter Block: (len=14)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_WAIT
0002 (00002) TPB$K_LOCK_READ (reserving) "EMPLOYEES" TPB$K_SHARED
SQL> show flags

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

In some environments, it may be possible to define RDMS$DEBUG_FLAGS_OUTPUT to the NL: device to discard this output.

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.3 Oracle RMU Errors Fixed

2.3.1 RMU /CONVERT From V7.1 to V7.1 Did Not Preserve Client Sequences

Bug 2417207

In Oracle Rdb 7.1, a conversion of a database from V7.1 to V7.1 did not preserve any client sequences defined in the database root file. This caused bugcheck dumps for SQL queries involving client sequences since the system table RDB$SEQUENCES referenced client sequences that were no longer in the database root file. Note that this problem only happened for database conversions where the database to be converted was already at the current V7.1 version and client sequences had been defined prior to the conversion.

The following example shows the problem where the convert from V7.1 to V7.1 with client sequences defined completed but then caused bugchecks for SQL queries involving client sequences.

```
$sql
SQL> attach 'filename DEVICE:[DIRECTORY]TESTDB.RDB';
SQL> create sequence EMPID start with 123;
SQL> commit;
SQL> exit
$ rmu/convert DEVICE:[DIRECTORY]TESTDB.RDB;
%RMU-I-RMUTXT_000, Executing RMU for Oracle Rdb V7.1-03
%RMU-I-LOGCONVRT, database root converted to current structure level
%RMU-W-NOCVTCOM, Database DEVICE:[DIRECTORY]TESTDB.RDB;1
is already at the current structure level.
$ sql
attach 'filename DEVICE[DIRECTORY]testdb';
select EMPID.nextval from rdb$database;
%RDMS-I-BUGCHKDMP, generating bugcheck dump file DEVICE:[DIRECTORY]
RDSBUGCHK.DMP;
%COSI-F-BUGCHECK, internal consistency failure
```

To avoid this problem, do not convert a database from Rdb V7.1 to Rdb V7.1 if client sequences have been defined for the database prior to the conversion. If this problem happens, do a full restore of the V7.1 database. If no database backup exists, contact Oracle Rdb support.

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

2.3.2 RMU/COPY and RMU/MOVE Did Not Preserve Database Client Sequences

Bug 2434332

RMU/COPY and RMU/MOVE did not preserve any client sequences defined in the original database root when creating a new database root. This caused bugchecks if SQL queries involving client sequences were made to the copied or moved database. If no client sequences were defined for the copied or moved database, this problem did not happen. This problem has been corrected and any client sequences defined in the original database root are preserved in the copied or moved database root.

The following example shows the problem when client sequences were defined for a database, the database root was then moved or copied, and SQL queries which involved client sequences were then made to the copied or moved database.

```
$sql
SQL> att 'filename mf_personnel';
SQL>create sequence EMPID start with 123;
SQL> select EMPID.nextval from rdb$database;
                  123
1 row selected
SQL> exit
$create/dir [.copy]
ALPHA4>RMU/COPY_DATABASE MF_PERSONNEL /DIRECTORY=DEVICE:[.COPY]
SQL> att 'filename [.copy]mf_personnel';
SQL>select EMPID.nextval from rdb$database;
%RDMS-I-BUGCHKDMP, generating bugcheck dump file
DEVICE:[DIRECTORY]RDSBUGCHK.DMP;
```

To avoid this problem, either move or copy the database without any client sequences defined or use RMU/BACKUP and RMU/RESTORE to move or copy the database.

This problem has been corrected in Oracle Rdb Release 7.1.0.3.

Chapter 3
Software Errors Fixed in Oracle Rdb Release 7.1.0.2

This chapter describes software errors that are fixed by Oracle Rdb Release 7.1.0.2.

3.1 Software Errors Fixed That Apply to All Interfaces

3.1.1 Zero Index Prefix Cardinality After Create Index

Bug 867890

Under certain conditions, index prefix cardinality stored for a newly−created sorted index was incorrect (zero). This could sometimes occur when a table already had rows stored in it. When the index prefix cardinalities are not stored (are zero), the query optimizer might choose poor query strategies resulting in slow response times.

The following is an example illustrating the problem. A table, TT, is created with two data rows. Next, a unique index, TT_U, is created on that table and the transaction is committed. The ensuing select statement lists the index segments and the index prefix cardinality stored for each segment. For index TT_U, which has three segments, there are two index prefixes: (1) the column S by itself, and (2) the column S with the column E. The example below shows that the index prefix cardinalities were zero both after the index creation was committed and also after a disconnect from the database had been performed.

```
SQL> create table tt (s char (4), e char (1), v int);
SQL> insert into tt values ('ABC', 'Z', 10000000);
1 row inserted
SQL> insert into tt values ('ABC', 'Z', 10000001);
1 row inserted
SQL> commit;
SQL>
SQL> create unique index tt_u on tt (s,e,v);
SQL> commit;
SQL>
SQL> select cast(rdb$field_name as char(1)) as col,
cont>        cast(rdb$field_position as tinyint) as pos,
cont>        cast(rdb$cardinality as tinyint) as pfx_card
cont>   from rdb$index_segments where rdb$index_name = 'TT_U';
 COL     POS   PFX_CARD
 S         1          0
 E         2          0
 V         3          0
3 rows selected
SQL> rollback;
SQL>
SQL> disconnect all;
SQL>
SQL> attach 'filename test.rdb';
SQL>
SQL> select cast(rdb$field_name as char(1)) as col,
cont>        cast(rdb$field_position as tinyint) as pos,
cont>        cast(rdb$cardinality as tinyint) as pfx_card
cont>   from rdb$index_segments where rdb$index_name = 'TT_U';
 COL     POS   PFX_CARD
 S         1          0
 E         2          0
 V         3          0
3 rows selected
SQL> rollback;
```

As a workaround, to correct this error following index creation, use the RMU utility to collect optimizer cardinality statistics for the problem index.

```
$ RMU /COLLECT OPTIMIZER_STATISTICS /STATISTIC=CARDINALITY TEST.RDB
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2. Now, index prefix cardinalities will be recorded for newly−created indexes as soon as the work is committed.

3.1.2 RDB−E−ARITH_EXCEPT Error From the Rdb Optimizer

Bug 1694309

When using workload statistics, it was possible that a query that joined several tables together would produce a divide by zero error.

The following example shows the result of trying to execute a query that exposed the problem.

```
%RDB−E−ARITH_EXCEPT, truncation of a numeric value at runtime
−SYSTEM−F−HPARITH, high performance arithmetic trap, Imask=00000000,
  Fmask=00000001, summary=04, PC=0000000000FBF748, PS=0000000B
−SYSTEM−F−FLTDIV, arithmetic trap, floating/decimal divide by zero at
  PC=0000000000FBF748, PS=0000000B
```

As a side effect of this problem, some queries could be inaccurately costed by the optimizer, which could lead to less than optimal retrieval strategies. The following simple example shows a query where the cardinality was inaccurately calculated from the workload statistics because of this problem.

```
SQL> set flags 'estimates'
SQL>  select * from t1, t2 where t1.f1=t2.f1;
Solutions tried 6
Solutions blocks created 4
Created solutions pruned 1
Cost of the chosen solution   1.5162601E+01
Cardinality of chosen solution   0.0000000E+00
~O: Workload statistics used
       T1.F1         T2.F1
           1             1
.
.
.
1000 rows selected
```

Oracle Rdb now correctly interprets NULL factors of 1.0 and 0.0 in workload statistics and therefore correctly calculates the cardinality of this example to 1000 rows.

The problem can be worked around using any of the following techniques:

- Ensuring that workload data does not have a null factor of exactly 0.0 or 1.0.
- Removing workload statistics.
- Ensuring that the table cardinalities are greater than 1 for all tables in the query.
- Use of the *OLD_COST_MODEL* debug flag.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.3 Page Locking Problems in Release 7.1.0 and Release 7.1.0.1

Bug 2042873

Oracle Rdb Release 7.1 introduced errors into the buffer page locking mechanisms that could cause excessive stalls or deadlocks.

The first problem was triggered when the Asynchronous Prefetch (APF) mechanism was used to fetch a buffer that contained only one page. In that situation, blocking ASTs for the page lock would be ignored. This was typically seen for buffers containing Space Area Management (SPAM) pages.

Regular user processes rarely read SPAM pages via APF, but the AIJ Log Recovery Server (LRS) will often use APF to read SPAM pages. Processes attempting to read the standby database while the LRS was in operation would sometimes see long stalls for SPAM page locks since the LRS was neglecting to process the blocking AST requests.

When not using Hot Standby, this problem may be avoided by disabling APF. However, it is not possible to disable APF for the LRS.

The second problem was seen when Global Buffers were enabled. In that situation, if one process read a buffer via the APF mechanism, and a second process wanted to access pages within the same buffer, the second process would not use the proper locking protocol to ensure that the first process was properly notified via the blocking AST mechanisms. This could lead to excessive stalls for page locks and deadlocks on page locks. This problem was quite noticeable when the LRS process needed to access a page being held by processes doing online access to the standby database. It was possible for the LRS to encounter so many lock conflicts that it could not process fast enough and would throttle activity on the master database.

To workaround this problem, global buffers may be disabled. This may, however, induce a substantial performance degradation in the application.

These problems have been corrected in Oracle Rdb Release 7.1.0.2.

3.1.4 Storage Area Default Size Increase

Bug 2151253

The storage area default size was 400 pages which was too small and always caused the area to be extended at least once during database creation. This default has been increased to 700 pages which is now just large enough to not require extending during database creation.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.5 Recovery Process Caused Excessive Snapshot File Growth

Bug 2033576

In Oracle Rdb Release 7.1.0.1, it was possible for the Database Recovery process (DBR) to excessively extend snapshot files, and perhaps fail with a bugcheck dump containing an error similar to the following:

```
***** Exception at 0017040C : PIO$EXTEND_STAREA + 0000097C
%RDMS-F-FILACCERR, error extending file DEV:[DIR]SNAPSHOT_FILE.SNP;
-SYSTEM-W-DEVICEFULL, device full; allocation failure
```

This would typically happen after a process had inserted many rows in the database and, before the transaction was committed, there was a system failure or the database was closed with the RMU/CLOSE/ABORT=DELPRC command. In that situation, the DBR would needlessly store before image entries of all of the inserted rows into the snapshot file(s), and it would not attempt to reuse any of the pages currently in the snapshot file(s).

This problem has been corrected in Oracle Rdb Release 7.1.0.2. After a node failure, the DBR will not attempt to write snapshot file entries when rolling back inserted rows.

3.1.6 Dynamic Optimization Estimation Incorrect for Ranked Indices

The dynamic optimization process was incorrectly calculating the cost of scanning indices of type *SORTED RANKED*.

In the following example, the table being queried has the numbers one to one thousand in both fields. The different ranges used should result in a different estimated cost. However in all cases the *ESTIM* phase computes the cost of scanning these indices as 680:

```
SQL> select * from t where f1 between 1 and 2 and f2 between 2 and 1000;
~S#0003
Leaf#01 FFirst T Card=1000
  BgrNdx1 T1 [1:1] Fan=17
  BgrNdx2 T2 [1:1] Fan=17
~E#0003.01(1) Estim   Ndx:Lev/Seps/DBKeys 1:34/0\680 2:34/0\680
~E#0003.01(1) BgrNdx1 EofData  DBKeys=2  Fetches=2+0  RecsOut=1 #Bufs=1
~E#0003.01(1) FgrNdx  FFirst   DBKeys=1  Fetches=0+1  RecsOut=1`ABA
~E#0003.01(1) Fin     Buf      DBKeys=2  Fetches=0+0  RecsOut=1
          F1            F2
           2             2
1 row selected
SQL> select * from t where f1 between 2 and 1000 and f2 between 1 and 2;
~S#0004
Leaf#01 FFirst T Card=1000
  BgrNdx1 T1 [1:1] Fan=17
  BgrNdx2 T2 [1:1] Fan=17
~E#0004.01(1) Estim   Ndx:Lev/Seps/DBKeys 1:34/0\680 2:34/0\680
~E#0004.01(1) BgrNdx1 EofData  DBKeys=999  Fetches=0+10  RecsOut=1 #Bufs=10
~E#0004.01(1) FgrNdx  FFirst   DBKeys=1  Fetches=0+11  RecsOut=1`ABA
~E#0004.01(1) Fin     Buf      DBKeys=999  Fetches=0+0  RecsOut=1
          F1            F2
           2             2
1 row selected
```

In the first example (query 3), the index T1 on field F1 is the correct index to use, as the key range is very small. In the second example (query 4), the index T2 on field F2 is the correct index to use. However, in both cases the indices are costed the same so no index reordering takes place.

Even in this small example, significantly more work is being performed in query 4 as can be observed from the I/O counts.

This problem is corrected in Oracle Rdb Release 7.1.0.2. Rdb now returns a costing from the ESTIM phase that reflects the different key value ranges for the query. The following example shows the corrected execution where query 4 reorders the index, resulting in significantly less I/O:

```
SQL> select * from t where f1 between 1 and 2 and f2 between 2 and 1000;
~S#0003
Leaf#01 FFirst T Card=1000
  BgrNdx1 T1 [1:1] Fan=17
  BgrNdx2 T2 [1:1] Fan=17
~E#0003.01(1) Estim   Ndx:Lev/Seps/DBKeys 1:1/1/1 2:35/0\681
~E#0003.01(1) BgrNdx1 EofData  DBKeys=2  Fetches=0+0  RecsOut=1 #Bufs=1
~E#0003.01(1) FgrNdx  FFirst   DBKeys=1  Fetches=0+1  RecsOut=1`ABA
~E#0003.01(1) Fin     Buf      DBKeys=2  Fetches=0+0  RecsOut=1
          F1            F2
           2             2
1 row selected
SQL> select * from t where f1 between 2 and 100 and f2 between 1 and 2;
~S#0004
Leaf#01 FFirst T Card=1000
  BgrNdx1 T1 [1:1] Fan=17
  BgrNdx2 T2 [1:1] Fan=17
~E#0004.01(1) Estim   Ndx:Lev/Seps/DBKeys 2:1/0\1 1:4/0\80
~E#0004.01(1) BgrNdx2 EofData  DBKeys=2  Fetches=0+0  RecsOut=1 #Bufs=1
~E#0004.01(1) FgrNdx  FFirst   DBKeys=1  Fetches=0+0  RecsOut=1`ABA
~E#0004.01(1) Fin     Buf      DBKeys=2  Fetches=0+0  RecsOut=1
          F1            F2
           2             2
1 row selected
```

The only workaround for this problem is to use indices of *TYPE IS SORTED* rather than of *TYPE IS SORTED RANKED*.

3.1.7 Bugchecks Truncating Table in Mixed−Format Area with Row Caches

Bug 1994856

In some cases, truncating a table where its data or indexes are stored in mixed format areas can result in a bugcheck. This bugcheck was caused by incorrectly processing "Row Cache Reserved" space on a database page.

This problem has been corrected in Oracle Rdb Release 7.1.0.2. The "Row Cache Reserved" space on the database page now causes the row in the cache to be correctly fetched and considered for deletion.

3.1.8 Fast Commit Checkpoints Do Not Always Advance

In previous releases of Oracle Rdb, a process' after−image journal checkpoint location would only be advanced when one of the following events occurred:

- A transaction ended (COMMIT/ROLLBACK)

- An RMU/CHECKPOINT command was issued
- A journal switchover occurred (multiple after image journals utilized)

If a process became idle for an extended period of time, the checkpoint location would not advance and could become quite old. This could be troublesome if a process failure occurred, since the old checkpoint location would require the database recovery process (DBR) to process all of the journal contents starting at the point of the last checkpoint location for the failed process. The DBR journal processing could take a considerable amount of time and all database processing would be frozen until the DBR completed.

This release introduces changes in the way the fast commit CHECKPOINT INTERVAL IS n SECONDS option is implemented. In the past, Oracle Rdb would only check to see if the time interval was exceeded at the end of a transaction. In this release, if the CHECKPOINT INTERVAL IS n SECONDS option has been specified, Oracle Rdb processes will periodically check to see if the checkpoint may be advanced, even if the process is in the middle of a transaction. After the specified number of seconds have elapsed, the current checkpoint location will be evaluated, and if any of the criteria specified for checkpoint advance (journal growth, transaction count, time) have been exceeded since the last checkpoint then the checkpoint will be advanced. Note that this means that a checkpoint can occur at any point in time, not just at the end of a transaction as was typically the case before.

In addition, if the COMMIT TO JOURNAL OPTIMIZATION option is not being used, and no updates have occurred during the number of seconds specified by the CHECKPOINT TIMED EVERY n SECONDS clause, then the journal checkpoint location will be cleared for that process. If the process does not have a current checkpoint and the process later terminates abnormally, that process will not require any after image journal processing by the DBR process.

The details of the new timer implementation is as follows. When a process first makes an update to the database, a timer is queued for CHECKPOINT INTERVAL IS n SECONDS in the future. When that timer expires, Oracle Rdb checks to see if the checkpoint should be advanced and then it queues another timer. The next time the timer expires, if no additional updates have been made, then the process flushes all modified database buffers to disk and the checkpoint location is reset. No further timers are queued until the process makes another update to the database. If updates have been made since that timer was queued then the process checks to see if the checkpoint should be advanced and then another timer is queued to check again later.

This enhancement also introduces a subtle change in the way that Oracle Rdb displays checkpoint locations. Previously, when a process had never checkpointed, the output from RMU/DUMP/USERS would display the following:

```
Active user with process ID 22005424
    Stream ID is 1
    Monitor ID is 1 (ALPHA4)
    Transaction ID is 10
    No transaction in progress
>>  Process has not yet checkpointed
    Last Process quiet−point was AIJ sequence 0
```

Since a process can now fluctuate between the states of having a checkpoint location or no longer having a checkpoint location, the output of RMU/DUMP/USERS now displays the following for a user that does not have a checkpoint location:

```
Active user with process ID 22005424
    Stream ID is 1
    Monitor ID is 1 (ALPHA4)
    Transaction ID is 10
```

```
    No transaction in progress
>>  Process has no current checkpoint
    Last Process quiet-point was AIJ sequence 0
```

In addition, the checkpoint line will only be displayed if the fast commit feature is enabled.

3.1.9 Monitor "Home" Directory

Bug 2205733

By default, the Oracle Rdb monitor (RDMMON) process inherits its default device and directory specification from the process that executed the RMU /MONITOR START command (typically when executing the RMONSTART71.COM procedure). This default device and directory specification is, in turn, inherited by the various server processes (such as DBR, RCS, ALS, and so forth).

It is, however, possible to "SET DEFAULT" to an invalid or non−existant directory. And it is also possible to delete a directory that might happen to be the default directory for another process. These types of events can cause database server processes to fail. The Oracle Rdb monitor process does attempt to detect an invalid directory specification, but it is not able to prevent an existing valid specification from becoming invalid (due, for example, to deleting the directory).

In order to provide additional control over the device and directory specification used by the monitor and to make the monitor and database servers more resilient to changes in the system, the monitor process has been enhanced in regards to its default directory.

The monitor attempts to translate a new logical name "RDM$MON_DIRECTORY". If this logical name exists and specifies a valid device and directory specification, the monitor process explicitly sets its default to that device and directory. If this logical name is not defined or does not specify a valid directory, the monitor attempts to use the default device and directory that it inherited from the process that started the monitor. If this does not specify a valid directory, the monitor attempts to use the SYS$MANAGER logical name. Finally, if this does not specify a valid directory, the monitor attempts to use the SYS$SYSTEM logical name.

Once the monitor has determined a valid default directory, it creates a temporary file in that directory. The monitor process leaves this file open until the monitor process is shut down. This open file prevents the directory from being deleted. Note, however, that it is still possible to cause database server processes to fail if you manually rename the directory structure such that the monitor's default device and directory specification is no longer valid. Oracle recommends that if you must rename any portion of the directory tree that is used as the monitor process's default directory, that you first shutdown the Oracle Rdb monitor.

3.1.10 Bugcheck When Using Persona With SQL/Services

Bug 2217920

A bugcheck would occur with the exception *%RDB−E−AUTH_UNTRUSTED, rdb_register_user must be called from a trusted user* when using the OCI protocol service to access an Oracle Rdb database with "SECURITY CHECKING IS EXTERNAL (PERSONA SUPPORT ENABLED)" and service owner account having only NETMBX and TMPMBX privileges.

A workaround is to give the service owner SYSPRV privilege.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.11 Query With Join Predicates on Leading Segments and Equality Filters Returns Wrong Results

Bug 2204152

The following query with join predicates on leading segments and equality filters should find 2 rows instead of 0 rows:

```
set flags 'strategy,detail';
SELECT T2.PRICE_AMT FROM T1, T2
  WHERE
    T2.CMP_NO = 1 AND
    T2.PROD_NO = 161255 AND
    T2.DIV_NO = 1 AND
    T2.CUST_NO = 10674 AND

    T1.CMP_NO = T2.CMP_NO AND
    T1.PROD_NO = T2.PROD_NO AND
    T1.DIV_NO = T2.DIV_NO AND
    T1.CUST_NO = T2.CUST_NO AND
    T1.QUOTE = 0
    ;
Tables:
  0 = T1
  1 = T2
Cross block of 2 entries
  Cross block entry 1
    Conjunct: 0.QUOTE = 0
    Conjunct: 0.DIV_NO = 1
    Conjunct: 0.CMP_NO = 1
    Index only retrieval of relation 0:T1
      Index name  T1_NDX [4:4]
        Keys: (0.DIV_NO = 1) AND (0.PROD_NO = 161255) AND
              (0.CUST_NO = 10674) AND
              (0.CMP_NO = 1.CMP_NO)  <== Note1: incorrect conjunct
  Cross block entry 2
    Leaf#01 FFirst 1:T2 Card=7843
      Bool: (1.CMP_NO = 1) AND (1.PROD_NO = 161255) AND (1.DIV_NO =
            1) AND (1.CUST_NO = 10674) AND (0.CMP_NO = 1.CMP_NO)
             AND (0.PROD_NO = 1.PROD_NO) AND (0.DIV_NO =
            1.DIV_NO) AND (0.CUST_NO = 1.CUST_NO)
      BgrNdx1 T2_NDX [2:2] Fan=13
        Keys: (0.CUST_NO = 1.CUST_NO) AND (0.PROD_NO = 1.PROD_NO)
        Bool: (1.CMP_NO = 1) AND (1.PROD_NO = 161255) AND (1.DIV_NO
               = 1) AND (1.CUST_NO = 10674)
0 rows selected

Note1: 1.CMP_NO references table T2 in the cross block entry 1
       where context 1 is not available yet.

Indexes on table T1:
T1_NDX                          with column CUST_NO
                                and column PROD_NO
                                and column DIV_NO
                                and column CMP_NO
                                and column QUOTE
```

```
Indexes on table T2:
T2_NDX                          with column CUST_NO
                                and column PROD_NO
                                and column START_DATE
                                and column DIV_NO
                                and column CMP_NO
```

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

1. The query joins two tables, T1 and T2, using all leading segments except the last one in T1_NDX index, e.g. T1.CMP_NO, T1.PROD_NO, T1.DIV_NO, T1.CUST_NO.
2. The last segment, T1.QUOTE of T1_NDX, is also used in the equality filter.
3. There is also an equality filter for each segment of T2_NDX used as a join predicate, e.g. T2.CMP_NO, T2.PROD_NO, T2.DIV_NO, T2.CUST_NO.

As a workaround, the query works if the SQL flag TRANSITIVITY is turned off.

```
set flags 'notransitivity, max_stability';

Tables:
  0 = T1
  1 = T2
Cross block of 2 entries
  Cross block entry 1
    Get     Retrieval by index of relation 1:T2
      Index name  T2_NDX [2:2]
        Keys: (1.CUST_NO = 10674) AND (1.PROD_NO = 161255)
        Bool: (1.CMP_NO = 1) AND (1.DIV_NO = 1)
  Cross block entry 2
    Conjunct: 0.QUOTE = 0
    Index only retrieval of relation 0:T1
      Index name  T1_NDX [4:4]
        Keys: (0.DIV_NO = 1.DIV_NO) AND (0.PROD_NO = 1.PROD_NO)
              AND (0.CUST_NO = 1.CUST_NO) AND
              (0.CMP_NO = 1.CMP_NO)             <== Note2
   T2.PRICE_AMT
          29.12
          29.12
2 rows selected

Note2: 1.CMP_NO references table T2 in the cross
       block entry 2 where context 1 is already made available in
       the cross block entry 1.
```

It also works if the optimizer statistics are collected by running RMU /COLLECT OPTIMIZER_STATISTICS on table T1.

```
Tables:
  0 = T1
  1 = T2
Cross block of 2 entries
  Cross block entry 1
    Conjunct: 0.QUOTE = 0
    Conjunct: 0.DIV_NO = 1
    Conjunct: 0.CMP_NO = 1
    Index only retrieval of relation 0:T1
      Index name  T1_NDX [5:5]    Direct lookup
```

```
        Keys: (0.CMP_NO = 1) AND
              (0.DIV_NO = 1) AND (0.PROD_NO = 161255) AND
              (0.CUST_NO = 10674) AND (0.QUOTE = 0) <== Note3
  Cross block entry 2
    Conjunct: (0.CMP_NO = 1.CMP_NO) AND (0.DIV_NO = 1.DIV_NO)
    Get     Retrieval by index of relation 1:T2
      Index name  T2_NDX [2:2]
        Keys: (0.CUST_NO = 1.CUST_NO) AND (0.PROD_NO = 1.PROD_NO)
        Bool: (1.CMP_NO = 1) AND (1.DIV_NO = 1) AND (1.CUST_NO =
              10674) AND (1.PROD_NO = 161255)
  CQD.PRICE_AMT              CQD.COST_AMT   CQD.DEAL_AMT
          29.12                     27.85           1.50
          29.12                     27.85           1.50
2 rows selected

Note3: Only table context 0 is referenced in the cross block entry 1.
       No reference is made to context 1 of table T2.
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.12 Query With Transitive Join Predicates and Non−equality Filter Bugchecks

Bug 2207963

The following query, which worked in previous releases, bugchecks in Oracle Rdb Release 7.1.0.1.

```
set flags 'strategy,detail';

select T1.PROC_CD, T1.SYS_CD,
       T2.RUN_NBR, T2.CALENDER, T2.PROC_COD
  From T1, T2, T3
  where     T2.SYS_CD    = T1.SYS_CD
        AND T2.PROC_CD   = T1.PROC_CD
        AND T2.SEQ_NBR   = T1.SEQ_NBR
        AND T2.CYCLE_CD  = T1.CYCLE_CD
        AND T2.PROFIL_CD = T1.PROFIL_CD

        AND T3.SYS_CD    = T2.SYS_CD
        AND T3.PROC_CD   = T2.PROC_CD
        AND T3.DAY_DATE  = T2.CALENDER
        AND T3.SEQ_NBR   = T2.SEQ_NBR
        AND T3.RUN_NBR   = T2.RUN_NBR
        AND T3.CYCLE_CD  = T2.CYCLE_CD
        AND T3.PROFIL_CD = T2.PROFIL_CD

        AND T1.SYS_CO = 'CPD'
        AND T2.CALENDER <= '15−JAN−2002'
   ;

Note: All the leading segments except the last one in T2_NDX index
      are used as join predicates.

Indexes on table T2:
T2_NDX                          with column SYS_CD
                                and column PROC_CD
                                and column CALENDER
                                and column SEQ_NBR
```

```
and column RUN_NBR
and column CYCLE_CD
and column PROFIL_CD
and column PROC_COD
```

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

1. The query joins 3 tables, T1, T2, and T3, where table T1 and T3 are joined via transitive selection predicates, such as "T1.col = T2.col and T2.col = T3.col".
2. Almost all of the leading segments, except the last one in the index T2_NDX, are referenced in the transitive predicates.
3. The filter predicate that references the 3rd leading segment, CALENDER, is a non−equality using "<=" operator.

As a workaround, the query works if the SQL flag TRANSITIVITY is turned off.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.13 Query With OR Predicates, Including Two Similar IS NULL Clauses, Returns Wrong Results

Bug 2177832

The following query with OR predicates, including two similar IS NULL clauses, should return 8 rows but instead returns 0 rows:

```
set flags 'strategy,detail';

select police.no_contra, police.cd_typol
from CMFasssoc asssoc, CMFpolice police, CMFassfam assfam, CMFserpol serpol
where
asssoc.statut <> 2
and asssoc.no_assure = 1670
and police.no_assure = asssoc.no_assure
and police.statut <> 2
and ((police.no_contra is null)
  or (police.no_contra is NOT null AND POLICE.CD_TYPOL <> 0)
    )
and assfam.no_assure = asssoc.no_assure
and assfam.statut <> 2
and serpol.no_assure = asssoc.no_assure
and serpol.no_police = police.no_police
and serpol.datd_mfac in (select max(serpol2.datd_mfac)
                         from CMFserpol serpol2
                         where serpol2.no_assure = asssoc.no_assure
                         and serpol2.no_police = police.no_police
                        )
;
Tables:
  0 = CMFASSSOC
  1 = CMFPOLICE
  2 = CMFADCLI2
  3 = CMFSERPOL
  4 = CMFSERPOL
Cross block of 4 entries
  Cross block entry 1
```

```
    Conjunct: 2.NO_CLI = 0.NO_ASSURE
    Match
      Outer loop
        Conjunct: 2.STATUT <> 2
        Conjunct: (2.TYP_CLI = 1) AND (2.CFCA_CLI2 = 1) AND (2.TYP_CLIRE = 2)
        Leaf#01 Sorted 2:CMFADCLI2 Card=2
          Bool: 2.NO_CLI = 1670
          FgrNdx  CMFADCLI2_I1 [3:3] Fan=9
            Keys: (2.NO_CLI = 1670) AND (2.TYP_CLI = 1) AND (2.CFCA_CLI2 = 1)
          BgrNdx1 CMFADCLI2_I2 [1:1] Fan=9
            Keys: 2.TYP_CLIRE = 2
            Bool: (2.NO_CLI = 1670) AND (2.TYP_CLI = 1) AND (2.CFCA_CLI2 = 1)
      Inner loop      (zig-zag)
        Conjunct: (0.STATUT <> 2) AND (0.NO_ASSURE = 1670)
        Get     Retrieval by index of relation 0:CMFASSSOC
          Index name  CMFASSSOC_I1 [1:1]
            Keys: 0.NO_ASSURE = 1670
  Cross block entry 2
    Conjunct: 2.STATUT <> 2
    Leaf#02 FFirst 1:CMFPOLICE Card=2
      Bool: (0.STATUT <> 2) AND (0.NO_ASSURE = 1670) AND (1.NO_ASSURE =
            0.NO_ASSURE) AND (2.NO_CLI = 0.NO_ASSURE)
      BgrNdx1 POLICE_H_IDX_1 [1:1] Fan=1
        Keys: 1.NO_ASSURE = 0.NO_ASSURE
        Bool: 1.NO_ASSURE = 1670
      BgrNdx2 CMFPOLICE_I2 [0:1,1:1] Fan=12
        Keys: r0: NOT MISSING (1.NO_CONTRA)
              r1: MISSING (1.NO_CONTRA)
  Cross block entry 3
    Aggregate: 0:MAX (4.DATD_MFAC)
    Conjunct: (1.STATUT <> 2) AND <error: missing expression> <== NOTE (1)
              AND (1.CD_TYPOL <> 0)
    Conjunct: 4.NO_POLICE = 1.NO_POLICE
    Get     Retrieval by index of relation 4:CMFSERPOL
      Index name  SERPOL_H_IDX_1 [1:1]
        Keys: 4.NO_ASSURE = 0.NO_ASSURE
        Bool: 4.NO_ASSURE = 1670
  Cross block entry 4
    Conjunct: (3.NO_POLICE = 1.NO_POLICE) AND (3.DATD_MFAC = <agg0>)
    Get     Retrieval by index of relation 3:CMFSERPOL
      Index name  SERPOL_H_IDX_1 [1:1]
        Keys: 3.NO_ASSURE = 0.NO_ASSURE
        Bool: 3.NO_ASSURE = 1670
0 rows selected

NOTE (1) : Error in the conjunct indicates some expression is missing.
           This is the cause of the problem.
```

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

1. The query joins 3 tables and one simple view.
2. The WHERE clause includes several join predicates, some filter predicates and an IN clause on a subquery.
3. One of the filter predicates contains an OR expression with similar IS NULL clauses on each branch.

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

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.14 Query Slows Down Using Full Index Scan [0:0]

Bug 1635351

A query that worked well in Oracle Rdb Release 7.0.1.2 became much slower in Oracle Rdb Release 7.0.6 using full index scan. Even if the customer uses the same outline as before, the performance does not improve. Here is the query:

```
select h.hnmei_id,
       h.hnmei_nm
from   pm_zumen_v p,
       zumen_v    z,
       hinmei_v   h
where  p.hinban = '000704419' and
       p.zuban  = z.zuban and
       z.teisei_kgo in ( select max(z1.teisei_kgo)
                           from zumen_v z1
                           where z.zuban = z1.zuban ) and
       z.zuban      = h.zuban and
       z.teisei_kgo = h.teisei_kgo
```

The Oracle Rdb Release 7.0.1.2 strategy chosen was the following:

```
Cross block of 4 entries
  Cross block entry 1
    Index only retrieval of relation PM_ZUMEN
      Index name  IDX_PM_ZUMEN_0 [1:1]
  Cross block entry 2
    Conjunct        Index only retrieval of relation ZUMEN
      Index name  IDX_ZUMEN_0 [1:1]
  Cross block entry 3
    Conjunct        Aggregate      Index only retrieval of relation ZUMEN
      Index name  IDX_ZUMEN_0 [2:2]  Min key lookup
  Cross block entry 4
    Index only retrieval of relation HINMEI
      Index name  IDX_HINMEI_0 [3:3]
0 rows selected
```

The Oracle Rdb Release 7.0.6 strategy chosen was the following:

```
Cross block of 3 entries
  Cross block entry 1
    Conjunct
    Match
      Outer loop
        Index only retrieval of relation ZUMEN
          Index name  IDX_ZUMEN_0 [0:0]          <-- full index scan
        Inner loop      (zig-zag)
          Aggregate      Index only retrieval of relation ZUMEN
            Index name  IDX_ZUMEN_0 [0:0]        <-- full index scan
  Cross block entry 2
    Conjunct        Index only retrieval of relation PM_ZUMEN
      Index name  IDX_PM_ZUMEN_0 [1:1]
  Cross block entry 3
    Index only retrieval of relation HINMEI
      Index name  IDX_HINMEI_0 [3:3]
0 rows selected
```

There is no known workaround for this problem. Even an outline that switches from match to cross strategy is unable to apply full index scan.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.15 Poor Choice of Indexes by Dynamic Optimizer

Bug 703558

A query that worked well in Oracle Rdb Release 6.0 took ten times longer to execute in Oracle Rdb Release 7.0. The problem was attributed to a poor choice of indexes used by the dynamic optimizer. Here is the query:

```
select a.ass_asset_code, a.ass_asset_name, i.tot_clients, i.tot_value
  from (select ass_asset_code,
               count (*) as tot_clients,
               sum(asset_value) as tot_value
        from investment
        where dlr_dealer_id starting with ''                    <-- note
          and ofc_office_id starting with '0119027BO01'         <-- note
          and adv_adviser_id starting with ''                   <-- note
          and cln_service_type starting with ''                 <-- note
        group by ass_asset_code) i,
       asset a
  where i.ass_asset_code = a.ass_asset_code
  order by a.ass_asset_code asc;
```

The WHERE clause includes these conditions:

```
    dlr_dealer_id starting with ''
    ofc_office_id starting with '0119027BO01'
    adv_adviser_id starting with ''
    cln_service_type starting with ''
```

The Oracle Rdb 6.0 strategy chosen was the following:

```
Conjunct
Match
  Outer loop
    Merge of 1 entries
      Merge block entry 1
      Aggregate       Sort
      Leaf#01 BgrOnly INVESTMENT Card=383229
        BgrNdx1 INVESTMENT_NDX_7 [1:1] Fan=14                   <-- note
        BgrNdx2 INVESTMENT_NDX_6 [1:1] Fan=14                   <-- note
        BgrNdx3 INVESTMENT_NDX_5 [1:1] Fan=14                   <-- note
        BgrNdx4 INVESTMENT_NDX_3 [1:1] Bool Fan=7               <-- note
  Inner loop      (zig-zag)
    Get     Retrieval by index of relation ASSET
      Index name  ASSET_NDX_2 [0:0]
```

Use of four background indexes makes sense because each has a different leading segment (column) matching one of the STARTING WITH clauses. The execution trace (not shown) indicates that the background scanned BgrNdx2 (INVESTMENT_NDX_6) to completion, but aborted all other scans due to reaching FtchLim. This also makes sense because the leading segment of this index is OFC_OFFICE_ID, which is the only column

for which a real value is provided in the STARTING WITH clause. In other words, Rdb is able to retrieve the necessary rows using index INVESTMENT_NDX_6 without having to do a full index scan.

The Oracle Rdb 7.0 strategy chosen was the following:

```
Conjunct
Match
  Outer loop
    Merge of 1 entries
      Merge block entry 1
      Aggregate       Sort
      Leaf#01 BgrOnly INVESTMENT Card=383229
        BgrNdx1 INVESTMENT_NDX_3 [1:1] Bool Fan=7              <-- note
        BgrNdx2 INVESTMENT_NDX_1 [1:1] Bool Fan=5              <-- note
  Inner loop      (zig-zag)
    Get     Retrieval by index of relation ASSET
      Index name  ASSET_NDX_2 [0:0]
```

Note that INVESTMENT_NDX_6 was not selected as a candidate index. This means that whichever index is chosen, a full index scan will have to be performed since the STARTING WITH clauses on these indexes have values of an empty string ('). The end result is that there is an order of magnitude more I/O for Oracle Rdb 7.0.

The new strategy after the fix is as follows.

```
Conjunct
Match
  Outer loop
    Merge of 1 entries
      Merge block entry 1
      Aggregate       Sort
      Leaf#01 BgrOnly INVESTMENT Card=383229
        BgrNdx1 INVESTMENT_NDX_1 [1:1] Bool Fan=5
        BgrNdx2 INVESTMENT_NDX_8 [1:1] Fan=14
        BgrNdx3 INVESTMENT_NDX_5 [1:1] Fan=14
        BgrNdx4 INVESTMENT_NDX_6 [1:1] Fan=14
        BgrNdx5 INVESTMENT_NDX_7 [1:1] Fan=14
        BgrNdx6 INVESTMENT_NDX_4 [1:1] Fan=10
  Inner loop      (zig-zag)
    Get     Retrieval by index of relation ASSET
      Index name  ASSET_NDX_2 [0:0]
```

As a workaround, a query outline can be used. However, in Oracle Rdb Release 7.0.1.2, the version under which the problem was reported, it was not possible to work around the problem by defining a query outline. That was a separate problem. A correction to allow a query outline to be used in this case became available in Oracle Rdb Release 7.0.2.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.16 UNION Query With Constant Column Returns Wrong Results

Bug 2231693

The following UNION query with constant column should return 1 row.

```
set flags 'strategy,detail';

create table t1 (art_no char(12), art_rev char(12));
create table t2  (art_no char(12), art_rev char(12));
insert into t1 values ('053 2021−120', ' ');
create view t1_view
  as select
  art_no, art_rev from t1;
create view t2_view
  as select
  art_no, art_rev from t2;

select v.art_no, v.art_rev
from (
      select adr.*, 'X' as RevType
      from t2_view adr
      union
      select a.*, ' ' as RevType
      from t1_view a
     ) v
where
    v.art_no = '053 2021−120' and
    RevType = ' ' ;

Tables:
  0 = T2
  1 = T1
Merge of 1 entries
  Merge block entry 1
  Reduce: <mapped field>, <mapped field>, <mapped field>
  Sort: <mapped field>(a), <mapped field>(a), <mapped field>(a)
  Conjunct: 'X' = ' '                  <== Note1
  Merge of 2 entries
    Merge block entry 1
    Conjunct: 0.ART_NO = '053 2021−120'
    Conjunct: 'X' = ' '
    Get     Retrieval sequentially of relation 0:T2
    Merge block entry 2
    Conjunct: 1.ART_NO = '053 2021−120'
    Conjunct: ' ' = ' '
    Get     Retrieval sequentially of relation 1:T1
0 rows selected
```

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

1. The query selects from the derived table of 2 unioned subselect queries which select all columns plus additional constant column from the simple view of each table.
2. The WHERE clause contains the equality predicate referencing the constant column of the unioned derived table.

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.17 Query With CAST Function Using Ranked Index Signals Exception Error

Bug 2235593

The following query with CAST function using ranked index signals an exception error:

```
create table t1 (y2k smallint, data_id char(11), proj_id char(6));
insert into t1 values (20,'20020202','2915');
create unique index t1_i1 on t1 (y2k, data_id) type is sorted ranked;
create unique index t1_i2 on t1 (proj_id) type is sorted;

select * from t1 where
        proj_id='2915' and
       cast (data_id as integer) = 20020202;
Leaf#01 FFirst 0:T1 Card=1
  Bool: (0.PROJ_ID = '2915') AND (CAST (0.DATA_ID AS INT) = 20020202)
  BgrNdx1 T1_I1 [0:0] Fan=12
    Bool: CAST (0.DATA_ID AS INT) = 20020202
  BgrNdx2 T1_I2 [1:1] Fan=16
    Keys: 0.PROJ_ID = '2915'
%RDB-E-ARITH_EXCEPT, truncation of a numeric value at runtime
-COSI-F-INPCONERR, input conversion error
```

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

1. The query contains a WHERE clause with 2 equality predicates. One of the predicates uses the CAST function.
2. The query uses dynamic optimizer strategy with 2 background indices, where the first one is a ranked index.
3. The first background index has 2 segments, where the second segment is referenced by the CAST function in the WHERE clause.

As a workaround, the query works if the dynamic optimizer is disabled by setting the SQL flag MAX_STABILITY.

```
select * from t1 where
        proj_id='2915' and
       cast (data_id as date vms) = '02-Feb-2002';
Leaf#01 FFirst 0:T1 Card=1
  Bool: (0.PROJ_ID = '2915') AND (CAST (0.DATA_ID AS DATE VMS) = '02-FEB-2002')
  BgrNdx1 T1_I1 [0:0] Fan=12
    Bool: CAST (0.DATA_ID AS DATE VMS) = '02-FEB-2002'
  BgrNdx2 T1_I2 [1:1] Fan=16
    Keys: 0.PROJ_ID = '2915'
    Y2K   DATA_ID       PROJ_ID
     20   20020202      2915
1 row selected
```

The query also works if t1_i1 is a non-ranked sorted index:

```
drop index t1_i1;
create unique index t1_i1 on t1 (y2k, data_id) type is sorted ;
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.18 External Functions Cannot Init, Reason 22

After upgrading to a newer OpenVMS version (e.g. V7.3), external functions that are "bind on server site" may fail to execute giving these errors:

```
%RDB-E-EXTFUN_FAIL, external routine failed to compile or execute successfully
-RDMS-E-EXTABORT, routine XXXXXX execution has been aborted
-RDMS-E-RTNSBC_INITERR, Cannot init. external routine server site executor;
reason 22
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.19 Bugchecks at PSII2SCANSTARTBBCSCAN

In prior releases of Oracle Rdb, it was possible that a query involving SORTED RANKED indexes could bugcheck when trying to establish a scan of a duplicate node.

```
***** Exception at 00A2EA30 : PSII2SCANSTARTBBCSCAN + 000004F8
%COSI-F-BUGCHECK, internal consistency failure
```

This condition only occurs with SORTED RANKED indexes where a sequence of inserts, updates and deletes of the same duplicate values force the production of an overflow duplicates node but subsequent deletes remove the duplicate entries that are on the primary index node for that duplicate value.

A possible workaround for this problem is to rebuild the sorted ranked index.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.20 Cursor on Ranked Index Returned too Many Records

Bug 2270786

A problem in the way the current record offset was determined for SORTED RANKED index duplicate entries may cause Oracle Rdb to return the same record twice on a table cursor fetch.

This problem would only occur given the following circumstances:

- A cursor is established on a table and strategy shows that a sorted ranked index will be used to retrieve the records.
- The cursor fetch returned the first duplicate record in a duplicate entry with exactly two duplicates.
- The same process with this cursor open inserts a new record into or removes another record from the same table.
- The insert or delete happened to update the same index node currently referenced by the cursor.

In this situation, Oracle Rdb must invalidate the current fetch scan and re−establish its currency. However, the currency was incorrectly set to the first duplicate in the current entry, hence returning this record a second time on the next fetch.

Workarounds for this problem include:

- Rebuilding the index may provide a temporary workaround for this problem.
- Change the processing of the records so as to not interleave fetches and inserts in the same process in this manner.
- Alternatively, rebuild the index as a normal SORTED index.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.21 Changed Default Behavior for Bitmapped Scan Optimization

Prior to this release of Oracle Rdb, bitmapped scan optimization was enabled by default when the dynamic optimizer found two or more sorted ranked indexes that could be used to satisify the query being optimized.

This default behavior has now been changed. Bitmapped scan optimizations must now be explicitly enabled.

Bitmapped scan optimization retrieval can be enabled using the debug flag 'BITMAPPED_SCAN'.

For example:

```
SQL> set flags 'BITMAPPED_SCAN';
```

This new behavior is in Oracle Rdb Release 7.1.0.2.

3.1.22 Bugcheck (ACCVIO) On Simple Select Statement

Bug 2298278

Some queries may generate bugchecks as shown in the following example.

```
SQL> SELECT NN, PTYPE, AMOUNT
cont> FROM TABLE1
cont> WHERE BNAME = 'ANYBODY' AND
cont> NN > DATE VMS '12-FEB-2001';
%RDMS-I-BUGCHKDMP, generating bugcheck dump file DISK1:[TEST]RDSBUGCHK.DMP;
```

The exception reported in the bugcheck dump file is:

```
***** Exception at 01F81EB8 : symbol not found
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
virtual address=000000004FFFFFD2, PC=0000000001F81EB8, PS=0000000B
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.23 Privileged User Bugcheck (ACCVIO)

Bug 2297264

A privileged user with no access granted to the database could receive an ACCVIO error and a bugcheck when executing actions outside of a transaction (for example, calling a stored procedure).

The following example shows the bugcheck exception report:

```
***** Exception at 00000004 : symbol not found
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual
    address=0000000000000004, PC=0000000000000004, PS=0000000B
```

A possible workaround is to grant access to the user.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.24 Bugchecks at DIOCCH$FETCH_SNAP_SEG + 00000594

Bug 1879372

In rare cases of relatively high system load with intensive access to cached records between read−write and read−only processes, it was possible for a read−only process to fail with an exception at DIOCCH$FETCH_SNAP_SEG + 00000594.

This bugcheck was due to incorrect memory access ordering. Read−only processes would sometimes get an incorrect snapshot page pointer and find that the snapshot page was not for the matching live page.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.1.25 Unresolved 2PC Transactions Rolled Back by RMU/RECOVER

Bug 2278911

When an RMU/RECOVER process completed processing the last journal specified, if the database was involved in a two−phase commit (2PC) transaction and the transaction was prepared but not yet committed (an "unresolved" transaction) when journal processing was complete, RMU/RECOVER would sometimes rollback the prepared transaction. Also, the "Current roll−forward sequence number" would be advanced to the next journal even though a transaction from the current journal was not completed.

This behavior was incorrect since unresolved transactions should be considered still active and must remain active until a commit or rollback record is found in the journal or the user explicitly instructs RMU/RECOVER to commit or abort the 2PC transaction. Advancing the "Current roll−forward sequence number" also allowed subsequent RMU/RECOVER commands to not require the journal(s) that contained the unresolved transaction. If the journal(s) containing the unresolved transaction was not applied again, the unresolved transaction would be lost.

When this situation occurred, output similar to the following would be observed from the RMU/RECOVER command:

```
%RMU-I-AIJACTIVE, 1 active transaction not yet committed or aborted
%RMU-I-LOGRECSTAT, transaction with TSN 0:143 is active
%RMU-I-AIJPREPARE, 1 of the active transactions prepared but not yet committed
or aborted
%RMU-I-AIJSUCCES, database recovery completed successfully
%RMU-I-AIJNXTSEQ, to continue this AIJ file recovery, the sequence number
needed will be 1
```

```
%RMU−I−LOGRECSTAT, transaction with TSN 0:143 rolled back
.
.
.
%RMU−I−AIJFNLSEQ, to start another AIJ file recovery, the sequence number
needed will be 1
```

Note that in this example the active transaction was rolled back even though it was not yet resolved. Also, the sequence number was advanced to the next journal even though the active transaction had not been resolved.

One situation where this could occur is when the prepare record was stored in one journal but the commit record was stored in the next journal. In that situation, the transaction could be lost if multiple RMU/RECOVER commands were used to recover the database. To prevent that from occurring, all available journals should be specified in a single RMU/RECOVER command. That is, there shouldn't be a separate RMU/RECOVER command issued for each journal; all journals must be applied by a single RMU/RECOVER command.

This problem has been corrected in Oracle Rdb Release 7.1.0.2. At the end of journal processing, if there is an unresolved transaction still active, the transaction will remain active and the "Current roll−forward sequence number" will not be advanced.

3.2 SQL Errors Fixed

3.2.1 Queries Ending in Reserved Words Fail to Execute in Dynamic SQL

Bug 2088594

If the final token of a query is a column whose name is a reserved word then the query may fail with SQL−F−PREMATURE_EOF. However, if extra syntax is added to the query it will work. Similarly, if the column is prefixed with the table name or correlation name (such as TT.POSITION) then the query succeeds.

The following example shows the problem using a dynamic SQL program. When the query is extended by adding an additional column to the ORDER BY clause, the query succeeds.

```
>> CREATE TABLE TT (AA INT, POSITION INT)
>> INSERT INTO TT (AA, POSITION) VALUES (1, 1)
>> INSERT INTO TT (AA, POSITION) VALUES (1, 2)
>> SELECT * FROM TT ORDER BY POSITION
error: −1...
error text:
%SQL−F−PREMATURE_EOF, Statement is syntactically incomplete
>> SELECT * FROM TT ORDER BY POSITION, AA
out: 0:        0
out: 1:        0
 0/AA: INTEGER:1
 1/POSITION: INTEGER:1
 0/AA: INTEGER:1
 1/POSITION: INTEGER:2
>> ROLLBACK
```

The problem in this case is that POSITION is valid starting syntax for the POSITION function. Dynamic SQL requests the next token which is expected to be the start of the function argument list. However, an exception is raised because dynamic SQL does not permit continuations of statements. Similar problems occur if column names such as TRIM and SUBSTRING are used.

If this query was executed by interactive SQL, then the terminating semicolon (;) would indicate that the builtin function was not being used and the name would then be treated as a column name.

To solve this problem, the next release of dynamic SQL will permit an optional terminating semicolon (;). If more tokens are requested (as in this problem case), an implicit semicolon will be provided by SQL and the failing syntax may succeed.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.2 SQL$MOD Compiler Does Not Recognize G_FLOAT with COBOL

Bug 1149572

COBOL on VAX only supported D_FLOAT and not G_FLOAT. G_FLOAT support was added on Alpha but SQL$MOD still gave a warning for it.

For example, suppose a SQL Module Language program for the COBOL language declared a procedure with a parameter called ":P_FLOATFLD" which is of type "FLOAT". In this case, if the program is compiled with a /G_FLOAT qualifier, SQL$MOD would flag the declaration as having an unsupported datatype as follows:

```
$SQL$MOD /G_FLOAT EXAMPLE_PROG.SQLMOD
        :P_FLOATFLD     FLOAT);
                        1
%SQL-W-LANUNSDTP, (1) COBOL does not support the data type for parameter
P_FLOATFLD
```

This program will now compile without warnings on OpenVMS Alpha. The warning still (appropriately) appears for OpenVMS VAX.

There is no workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.3 Unexpected UNSDTPCVT Error Reported for NULL in UNION Statement

In Oracle Rdb Release 7.1.0.1, data type checking for UNION was improved to better support character set assignments. However, this change introduced a problem with NULL expression processing as shown in the following example:

```
SQL> select NULL as literal_suffix
cont>  from rdb$database
cont> union
cont> select '''' as literal_suffix
cont>  from rdb$database;
%SQL-F-UNSDTPCVT, Unsupported data type conversion
SQL>
SQL> select NULL
cont>  from rdb$database
cont> union
cont> select ''''
cont>  from rdb$database;
%SQL-F-UNSDTPCVT, Unsupported data type conversion
SQL>
SQL> select NULL
cont>  from rdb$database
cont> union
cont> select '''' as literal_suffix
cont>  from rdb$database;
%SQL-F-UNSDTPCVT, Unsupported data type conversion
```

SQL is now trying to process the special character set for the NULL and reporting this error: %SQL−F−UNSDTPCVT, Unsupported data type conversion.

The workaround is to reverse the SELECT statements in the UNION clause so that the NULL expression is processed last.

```
SQL> select '''' as literal_suffix
cont>  from rdb$database
cont> union
cont> select NULL as literal_suffix
cont>  from rdb$database;
 LITERAL_SUFFIX
 '
 NULL
2 rows selected
SQL>
SQL> select ''''
cont>  from rdb$database
cont> union
cont> select NULL as literal_suffix
cont>  from rdb$database;

 '
 NULL
2 rows selected
SQL>
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.4 Precompiled SQL Does Not Recognize a C Function With a Struct Return Type

Bug 1274182

If a C function's return type was a struct type, then Precompiled SQL didn't recognize it as a function. Instead it processed it as a variable and also threw away everything up to the next semicolon.

The following example shows a C function ("my_program") which SQL$PRE didn't recognize as a function because of its return type:

```
    struct my_struct
    {
    int rcn;
    } my_struct;

    EXEC SQL DECLARE ALIAS FOR FILENAME mf_personnel;

    struct my_struct my_program(char *first_name, char *last_name)
    {
      char lcl_last_name[256];
      EXEC SQL SELECT last_name INTO :lcl_last_name FROM employees
                     WHERE last_name  = :last_name
                       AND first_name = :first_name;
      return my_struct;
    }
```

The following interactive session shows the errors generated by SQL$PRE because it does not recognize "my_program" as a function. Since it ignores everything until the next semicolon, it doesn't recognize the function parameters or the host variable "lcl_last_name".

```
POLLUX> SQL$PRE/CC test_program.sc;
                  WHERE last_name  = :last_name
```

```
                                        1
%SQL-F-HVNOTDECL, (1) Host variable last_name was not declared
  EXEC SQL SELECT last_name INTO :lcl_last_name FROM employees
                                      1
%SQL-F-HVNOTDECL, (1) Host variable lcl_last_name was not declared
POLLUX>
```

The problem can be avoided by declaring the struct in a typedef and then using the resulting user defined type as the return type of the function.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.5 CREATE INDEX Placing Keys in Wrong Partition

Bug 2217239

Recently a problem has been found with the CREATE INDEX statement. This applies to the RDO DEFINE INDEX statement as well as the implicit CREATE INDEX performed by the SQL and RDO IMPORT statements.

If an index is partitioned on a single CHAR or VARCHAR column and that column is longer than 8 octets, then CREATE INDEX may place the index keys on the wrong partition when processing data currently in the table.

The Oracle Rdb 7.1 index scan optimization aborts the scan after the partition end is reached and so does not find the misplaced index keys. This will result in incorrect query results.

There is no known workaround for this problem. Please note that once the corrected version of Oracle Rdb is installed, the affected indices should be dropped and recreated.

This problem is not present in indices with the following characteristics:

- the index is not partitioned,
- has more than one column for the partitioning key (i.e. more than one column listed in the USING clause),
- has a data type other than CHAR or VARCHAR,
- is of type CHAR or VARCHAR with a length less than or equal to 8 octets,
- uses DESC or MAPPING VALUES clauses,
- based on a column with COLLATING SEQUENCE,
- any index created with Oracle Rdb 7.0 and present in the database when converted with RMU/CONVERT

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.6 ALTER INDEX ... TRUNCATE PARTITION Results in Bad Query Results

Bug 2206069

Recently a problem has been found with the ALTER INDEX ... TRUNCATE PARTITION statement.

The TRUNCATE PARTITION statement was not correctly setting the index to build−pending state and therefore the optimizer was erroneously using the partial index for data retrieval, which could result in incorrect query results.

If the index has been altered to MAINTENANCE IS DISABLED, then the TRUNCATE PARTITION statement can successfully be used to truncate parts of the index. In this case, the optimizer will not use this disabled index for query solutions.

As a workaround for this problem, use ALTER INDEX ... TRUNCATE ALL PARTITIONS instead of ALTER INDEX ... TRUNCATE PARTITION for each partition. Alternately, use ALTER INDEX ... MAINTENANCE IS DISABLED before using ALTER INDEX ... TRUNCATE PARTITION for each partition.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.7 ALTER INDEX ... BUILD ALL PARTITIONS Not Writing Back SORTED Index Root Dbkeys

Bugs 2195771 and 2199897

Recently, problems have been found with ALTER INDEX ... BUILD ALL PARTITIONS and ALTER INDEX ... REBUILD ALL PARTITIONS statements when used with SORTED or SORTED RANKED indices. Please do not use these statements until Oracle Rdb Release 7.1.0.2 or later has been installed.

The partition root dbkey for SORTED indices (both ranked and non−ranked) are not refreshed after the ALTER INDEX statement completes.

If the index has been altered with MAINTENANCE IS DISABLED, then the resulting index will be incomplete and may lead to incorrect query results or bugchecks.

If the index was processed with ALTER INDEX ... TRUNCATE ALL PARTITIONS, then the resulting index appears to be empty and may also lead to incorrect query results.

The REBUILD ALL operation causes the logical area for the index to be implicitly truncated. Queries may appear to function correctly but the pages are now marked for reuse and future updates will corrupt the structure of the sorted index as these B−tree nodes are overwritten.

This is not a problem for HASHED indices since these types of indices do not require the update of RDB$INDICES and RDB$STORAGE_MAP_AREAS tables with the root dbkeys.

As a workaround for this problem, the ALTER INDEX ... BUILD PARTITION and ALTER INDEX ... REBUILD PARTITION statements, which operate on just one partition, do correctly write back the root dbkeys and can be used as an alterative.

Use DROP INDEX and CREATE INDEX to rebuild the index structure.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.8 IMPORT Fails With INVIDXATTR Error for Hashed Indexes

Bug 2211328

If a database was exported, imported, exported again and finally imported a second time, the second import, under certain conditions, failed with an INVIDXATTR error for one or more hashed indexes.

This problem has been corrected in Oracle Rdb Release 7.1.0.2. The SQL IMPORT command no longer fails with an INVIDXATTR error.

3.2.9 DDL Statements Generated Unexpected Runtime Errors

In previous releases of Oracle Rdb, it was possible for DDL (data definition language) statements embedded in the SQL precompiler source (EXEC SQL) or in a SQL module language procedure to generate unexpected errors at run time. This problem only occurred when the quote character (') had to be doubled when included in a string literal.

Consider this CREATE TABLE example embedded in a C source module:

```
void sql_signal ();
main()
{
int SQLCODE = 0;
exec sql
    declare alias filename 'MF_PERSONNEL';
exec sql
    create table my_table1
        (name_q char(10) default '''');
if (SQLCODE != 0)
    sql_signal ();
exec sql rollback;
if (SQLCODE != 0)
    sql_signal ();
}
```

When this application is executed, the following error is reported:

```
%SQL-F-UNTSTR, Unterminated string found
```

The problem occurs because all DDL statements (such as CREATE TABLE) are processed as Dynamic statements by SQL module language and the SQL precompiler. The saved version of the CREATE TABLE statement is rewritten without processing the quoting character (') correctly.

In most cases, this problem would cause SQL-F-SYNTAX_ERR or %SQL-F-UNTSTR exceptions, but in some cases two mismatched quotes may have unexpectedly captured syntax and the statement may have executed correctly. See the following TRACE statement in a stored procedure for instance:

```
TRACE '''' || R.RDB$FIELD_NAME || '''';
```

This statement was saved as:

```
TRACE ''' || R . RDB$FIELD_NAME || ''';
```

which caused Trace to display the following text:

```
~Xt: ' || R . RDB$FIELD_NAME || '
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2. SQL now encodes the quoted string correctly for use with dynamic SQL. Any applications that suffer from this problem must be recompiled using the corrected version of Rdb.

3.2.10 INSERT Cursor on a Derived Table Would Bugcheck

In prior releases of Oracle Rdb, SQL did not prevent a derived table from being used as the target for an INSERT cursor. While the DECLARE and OPEN for the cursor apparently succeeded, attempts to use the cursor would generate a bugcheck as shown below.

```
SQL> declare ONE insert only table cursor
cont> for select EMPLOYEE_ID
cont>     from (select * from EMPLOYEES) as E;
SQL> OPEN ONE;
SQL> INSERT INTO CURSOR ONE VALUES ('00000');
%SQL-I-BUGCHKDMP, generating bugcheck dump file DISK1:[TESTING]SQLBUGCHK.DMP;
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual address=00000024,
PC=00239A72, PSL=03C00005
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2. SQL now issues an error when the DECLARE CURSOR is detected.

```
SQL> declare ONE insert only table cursor
cont> for select EMPLOYEE_ID
cont>     from (select * from EMPLOYEES) as E;
%SQL-F-NOUNION, UNION or derived table not valid in an INSERT or LIST CURSOR
```

3.2.11 CREATE TABLE Generates WISH_LIST for NULL Clause

Oracle Rdb Release 7.1.0.1 did not correctly support the NULL clause in the CREATE TABLE statement that was introduced in Oracle Rdb Release 7.1.0. This clause was supported for Oracle RDBMS compatibility and should have been ignored by Rdb. However, this clause generated the error as shown in the following example:

```
SQL> create table supplier (
cont>      suppid int not null,
cont>      name varchar(80) null);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-E-WISH_LIST, feature not implemented yet
```

The workaround is to remove the NULL clause from the CREATE TABLE statement.

This problem has been corrected in Oracle Rdb Release 7.1.0.2. The WISH_LIST error is no longer produced by Rdb.

3.2.12 Use of Synonyms Resulted in an Incorrect Query of System Tables

Oracle Rdb Release 7.1.0.1 did not correctly support the use of synonyms when querying the system tables. Specifically, when SQL needed to perform a query of the Oracle Rdb system tables, the synonym name was used for the query. This behavior is incorrect. SQL should use the value of the synonym, not the synonym itself.

One error which illustrates this incorrect behavior is included below. When processing the column definition, SQL tries to fetch the DEFAULT and raises an exception.

```
create domain id_dom int;
create table tbl1 (id id_dom primary key deferrable, text char (5));
create table tbl2 (id id_dom, text char (7));
create synonym t1 for table tbl1;
create synonym t2 for table tbl2;
alter table t2
    alter column id
        constraint authentic_id references tbl1 (id) deferrable;
%RDB-F-BAD_SEGSTR_ID, invalid segmented string identifier
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2. When querying the system tables, SQL correctly uses the value of the synonym.

3.2.13 SQL Query Bugchecks at SQL$$GET_QUEUE_WALK

Bug 2245763

In prior releases of Oracle Rdb, it was possible that some queries involving UNION and functions that returned VARCHAR would bugcheck when using ORACLE LEVEL1 dialect.

```
***** Exception at 002A5844 : SQL$$GET_QUEUE_WALK + 00000244
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
virtual address=0000000000000080, PC=00000000002A5844, PS=0000001B
```

The following example shows this problem.

```
SQL> set dialect 'oracle level1';
SQL>
SQL> create module MMM
cont>     language SQL
cont>     function rtrim (in :a varchar (200), in: c varchar (200))
cont>     returns varchar(200);
cont>     return trim (both :c from :a);
cont> end module;
SQL>
SQL> select ' '
cont> from employees e
cont> union
cont> select rtrim(e.first_name,' ')
cont> from employees e;
%SQL-I-BUGCHKDMP, generating bugcheck dump file DISK1:[TEST_DB]SQLBUGCHK.DMP;
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
virtual address=0000000000000080, PC=00000000002A5844, PS=0000001B
SQL>
```

This problem was caused by erroneous processing of the implicit CASE expression wrapped around the function call to produce Oracle RDBMS language semantics for zero length strings which are considered equivalent to NULL.

A workaround would be to use SET DIALECT 'SQL92' before executing this query. In this dialect, no special zero length string handling is required.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.14 SQL Query Bugchecks at SQL$$GET_QUEUE_WALK

Bug 2272808

In prior releases of Oracle Rdb, it was possible that some queries involving UNION, COALESCE (or NVL) builtin functions would bugcheck.

```
***** Exception at 003363D0 : SQL$$GET_QUEUE_WALK + 00000340
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
virtual address=0000000000000080, PC=00000000003363D0, PS=0000001B
```

The following example shows this problem.

```
SQL> create module MMM
cont>     language SQL
cont>     function fixstr (in :id integer,
cont>                 in :a char (20),
cont>                 in :b char (20),
cont>                 in :c char (20))
cont>     returns char(20);
cont>     return NULL;
cont> end module;
SQL>
SQL> select fixstr (1, last_name, first_name, middle_initial) as nm
cont>  from employees
cont>  where employee_id = '00164'
cont> union all
cont> select cast(coalesce(postal_code,
cont>                 fixstr (1, last_name, first_name, middle_initial)
cont>               ) as char(20)) as nm
cont>  from employees
cont>  where employee_id = '00164';
%RDMS-I-BUGCHKDMP, generating bugcheck dump file DISK1:[TEST_DB]SQLBUGCHK.DMP;
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
virtual address=0000000000000080, PC=00000000003363D0, PS=0000001B
```

This problem was caused by erroneous processing of the COALESCE expression wrapped around the function call in the second leg of the UNION clause.

Note

The NVL function is a synonym for COALESCE.

A workaround would be to rewrite the COALESCE as a searched case expression. *COALESCE (a, b, ..., z)* is equivalent to:

```
case
    when a is not NULL then a
    when b is not NULL then b
    ...
    else z
end
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.15 Multistatement Procedures Used with Connections Resulted in %RDB−E−OBSOLETE_METADA Error Message

Bug 1879521

In prior releases of Oracle Rdb, there was a problem with multiple connections and the use of multistatement procedures. Specifically, Oracle Rdb requires a special internal module to be set up for multistatement procedures. In the case of two or more connections calling the same multistatement procedure, the module setup was not done for the second connection. This was incorrect behavior and resulted in the following error message:

```
%RDB−E−OBSOLETE_METADA, request references metadata objects that no longer
exist
```

The correct behavior is to insure that the module setup is performed when a database switch occurs for the first time.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.2.16 Privileges Not Honored For SET TRANSACTION

Bug 1668270

Oracle Rdb was not reverting to the privilege settings of the SQL/Services service owner for commands such as SET TRANSACTION ... RESERVING, CREATE, ALTER and DROP.

The following example uses SQL*Plus and SQL*net for Rdb to execute a query and shows this behaviour.

```
SQL> set transaction read write reserving employees for protected write;
set transaction read write reserving employees for protected write;
*
ERROR at line 1:
ORA−01031: insufficient privileges
```

In fact, the current user is granted access to the EMPLOYEES table, but the service owner is not. A workaround is to give the service owner the required privileges.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3 Oracle RMU Errors Fixed

3.3.1 RMU Fails to Perform OPTIMIZER_STATISTICS Actions on Some Databases

In prior versions of Rdb, attempts to use RMU/SHOW OPTIMIZER_STATISTICS, RMU/COLLECT OPTIMIZER_STATISTICS, and related commands would fail if the default database character set was not DEC_MCS.

The following example shows the problem for a DEC_KANJI database.

```
$ RMU /SHOW OPTIMIZER_STATISTICS DISK1:[TESTING]SAMPLE.RDB
%RDB-F-CONVERT_ERROR, invalid or unsupported data conversion
-RDMS-E-CSETBADCOMPARE, incompatible character sets prohibit the requested
comparison
%RMU-F-FATALRDB, Fatal error while accessing Oracle Rdb.
%RMU-F-FTL_SHOW, Fatal error for SHOW operation at 29-OCT-2001 16:31:20.59

$ RMU /COLLECT OPTIMIZER_STATISTICS DISK1:[TESTING]SAMPLE.RDB
%RDB-F-CONVERT_ERROR, invalid or unsupported data conversion
-RDMS-E-CSETBADCOMPARE, incompatible character sets prohibit the requested
comparison
%RMU-F-FATALRDB, Fatal error while accessing Oracle Rdb.
%RMU-F-FTL_ANA, Fatal error for ANALYZE operation at 29-OCT-2001 16:31:36.12
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.2 RMU/CONVERT Fails to Correctly Define the RDB$WORKLOAD Table

When a database is converted to Rdb 7.1 and the optional system table RDB$WORKLOAD is present, Rdb fails to correctly define the metadata for this table and SQL is unable to see the data type for the RDB$NULL_FACTOR column.

The collection and utilization of workload data is unaffected by this problem. Only SQL applications are affected.

The following is an example of a database incorrectly converted from Rdb 7.0 to Rdb 7.1:

```
SQL> show table rdb$workload
Information for table RDB$WORKLOAD

Columns for table RDB$WORKLOAD:
Column Name                    Data Type        Domain
-----------                    ---------        ------
RDB$CREATED                    DATE VMS
RDB$LAST_ALTERED               DATE VMS
RDB$DUPLICITY_FACTOR           BIGINT(7)
RDB$NULL_FACTOR                Data type: 0
RDB$RELATION_ID                INTEGER
RDB$FLAGS                      INTEGER
RDB$FIELD_GROUP                CHAR(31)
```

```
RDB$SECURITY_CLASS              CHAR(20)
```

The RDB$NULL_FACTOR datatype is incorrectly interpreted. This will result in the following problem:

```
SQL> select rdb$null_factor from rdb$workload;
%SQL-F-FLDNOTCRS, Column RDB$NULL_FACTOR was not found in the tables in current
scope
```

A workaround for this problem is to have a sufficiently privileged user execute the following SQL command, commit, and then have applications that use this column DISCONNECT and reattach to the database.

```
SQL> update rdb$relation_fields set rdb$field_source='RDB$SCALED_COUNTER'
cont> where rdb$field_source='RDB$PROBABILITY';
```

This problem is corrected in Oracle Rdb Release 7.1.0.2. Rdb now correctly defines the RDB$WORKLOAD table during the RMU/CONVERT.

3.3.3 RMU Tape Density Problems Starting With OpenVMS V7.2−1

Bugs 1362656 and 1432269

Starting with Compaq OpenVMS V7.2−1, there were density problems for RMU commands that allow tape density values to be specified with the /DENSITY qualifier: RMU/BACKUP, RMU/BACKUP/AFTER_JOURNAL and RMU/OPTIMIZE_AIJ. These problems resulted in one of the following tape density related errors being returned when density values which were correct were specified. These values worked when specified in RMU commands prior to OpenVMS V7.2−1. The problems occurred with tape cartridges initialized to the new OpenVMS V7.2−1 MTD compaction values.

```
%RMU-E-DENSITY, TAPE_DEVICE:[000000]DATABASE.BCK; does not support specified
  density
%RMU-E-POSITERR, error positioning TAPE_DEVICE:
```

These problems resulted from problems in OpenVMS tape device drivers which were enhanced to handle the new MTD (multiple tape density) values introduced in OpenVMS V7.2−1. These problems caused the device drivers to incorrectly handle the existing tape density codes used prior to OpenVMS V7.2−1. These problems exist in VMS (some have been corrected) and cannot be fixed by RMU. However, RMU has been changed to avoid this problem by allowing the new MTD density codes to be specified by the /DENSITY command using the following syntax.

```
/DENSITY=(new_density_value,[NO]COMPACTION)
```

The existing density values can continue to be specified using the same syntax as before.

```
/DENSITY=existing_density_value
```

Please see the New Feature documentation on this enhancement for a full description (Section 5.1.2).

The following example shows the error returned when a valid density code was specified for a tape device with OpenVMS V7.2−1.

```
$RMU /BACKUP /DENSITY=70000 /REWIND /LABEL=(LABEL1,LABEL2) -
```

```
MF_PERSONNEL TAPE1:MFP.BCK, TAPE2:
%RMU-E-POSITERR, error positioning TAPE1:
```

This problem could sometimes be avoided by initializing the tape with OpenVMS V7.2−1 commands and not setting the density in the RMU command.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.4 RMU/VERIFY/ROOT Incorrectly Reports RMU−E−BADAIJPN and/or RMU−E−AIJNOTFND

Previously, it was possible for the RMU/VERIFY/ROOT command to incorrectly attempt to access a non−existant after−image journal file. This problem was caused by an incorrect bounds check that resulted in one additional, non−existant, internal data structure being used. In very rare cases, this data structure appeared to contain an incorrect (or blank) name of an after−image journal file.

For example, the following error might be displayed (note the two spaces between "file" and "not" in the second message; this is where the filename would typically be displayed − in this case the name was blank):

```
$ RMU /VERIFY /ROOT THUNDER.RDB
%RMU-E-BADAIJPN, There is no name associated with AIJ entry 1.
%RMU-E-AIJNOTFND, expected after-image file  not found
%RMU-W-ROOERRORS, 1 error encountered in root verification
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2. The RMU/VERIFY utility now only checks the valid internal data structure for after−image journal files.

3.3.5 RMU/CONVERT Problem With Database Wide Default Collating Sequence

Bug 2181768

There is a problem in Oracle Rdb RMU Release 7.1.0.1 and earlier 7.1 versions where, if a database with a database−wide default collating sequence defined is converted from an earlier version to Oracle Rdb Release 7.1, the database is corrupted and unusable. This is because a collating sequence name value is not inserted in the system field RDBVMS$COLLATION_NAME in the system table RDB$FIELD_VERSIONS for system fields added or modified by the RMU/CONVERT. The error RDMS−F−UNLIKECOLL is returned by SQL in queries involving system or user fields since the RDB$FIELD_VERSIONS system table gets referenced as part of the processing of the SQL query. On a SHOW TABLE, a %COSI−F−FILACCERR error will occur.

The following example shows an error returned from SQL when a database with a system wide collating sequence is accessed in SQL after it has been converted to Rdb Release 7.1.

In a prior Rdb version:

```
SQL> CREATE DATABASE ... TESTDB ... COLLATING SEQUENCE GERMAN GERMAN ...
```

In Rdb Release 7.1:

```
$RMU /CONVERT TESTDB.RDB
%RMU-I-CVTCOMSUC, CONVERT COMMITED for DUA0:[DB]TESTDB.RDB;1
```

```
SQL> SELECT * FROM TABLE1;
%RDB-F-CONVERT_ERROR, invalid or unsupported data conversion
-RDMS-F-UNLIKECOLL, fields of unlike collating sequence may not be
compared
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.6 RMU/BACKUP to Tape Could Hang and Not Finish

Bug 2136496

There is a problem in Oracle Rdb RMU Release 7.1.0.1 and earlier 7.1 versions where an RMU/BACKUP to tape can hang and not finish. This is most likely to happen if the backup is to a single tape drive or in cases where one storage area is still being read and written to tape when the other storage areas have already finished. This occurs because of a scheduling problem which causes a writer thread to assume its reader threads have finished when they are still active. This causes a deadlock situation where the writer thread keeps looping waiting for its reader threads to release their resources while the reader threads are waiting for a response from the writer thread.

A workaround for this problem is to use /READER_THREAD_RATIO=0 or READER_THREAD_RATIO=1 but this will cause the backup to take more time.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.7 RMU/BACKUP or RESTORE Bugcheck on Prompt to Mount a Tape Volume

There is a problem in Oracle Rdb RMU Release 7.1.0.1 and earlier 7.1 versions where, if during an RMU/BACKUP or RMU/RESTORE a tape is not ready on a drive as expected and either the RMU−I−READYREAD or RMU−I−READYWRITE prompt is output, an access violation will occur if the user presses return or inputs any value and presses return. The only way to avoid this is to have a tape ready on the drive so that this prompt does not need to be output.

The following shows an example of the prompt that caused this bugcheck.

```
$RMU /BACKUP /REWIND /LABEL=(LABEL1,LABEL2) TESTDB.RDB −
  tapedevice1:TESTDB.BCK, tapedevice2:

RMU-I-READYWRITE, mount volume 3 label LABEL03 on tapedevice1: for writing
Press return when ready:
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.8 RMU/BACKUP Prompt to Initialize Tape Label Created Incorrect Label

If a tape was mounted by RMU/BACKUP to tape which had an unexpected label and the user was prompted to either initialize the tape to a default label generated by RMU/BACKUP or to specify a label in the response to the prompt, an invalid label was generated. Now the default label or the label specified by the user when he

responds to the prompt is correctly used to label the tape.

This problem happens even though the user response to the prompt is one of the following valid responses:

```
INITIALIZE

(the above response allows RMU/BACKUP to generate a valid default label based
 on the /LABEL qualifier)

INITIALIZE LABEL1

INITIALIZE AS LABEL1
```

The following example shows that even though the user responded to the prompt with a valid label, the response was ignored.

```
$RMU /BACKUP /LOG /REWIND /LABEL=(LABEL1,LABEL2) -
 MF_PERSONNEL.RDB TAPE1:MF_PERSONNEL.BCK, TAPE2:

%MOUNT-I-MOUNTED, LABEL1 mounted on _TAPE1: (HJ50AC)
%RMU-I-WRNGLBL, Tape on _TAPE1 was incorrectly labeled. Expected LABEL1 -
Found XXX
%RMU-I-TAPEDISPW, Specify tape disposition for _TAPE1 (QUIT,INITIALIZE,
RETRY,UNLOAD)
RMU> INITIALIZE AS LABEL1
%MOUNT-I-MOUNTED, .`... mounted on _TAPE1: (HJ50AC)
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.9 RMU/RECLAIM Returns ACCVIO and Bugchecks at RMU_CLEANUP + 00000100

Bug 2232308

The RMU/RECLAIM command would often fail with an ACCVIO fatal error and a bugcheck with an exception at RMU_CLEANUP + 00000100. This failure occurred during the shutdown and exit phase of the RMU operation. The following example shows this problem:

```
$ RMU /RECLAIM /AREA=BAR FOO
%SYSTEM-F-ACCVIO, access violation, reason mask=00,
 virtual address=000000000000014C, PC=00000000003831C0, PS=0000001B
%RMU-F-FATALOSI, Fatal error from the Operating System Interface.
%RMU-I-BUGCHKDMP, generating bugcheck dump file DGA0:[ME]RMUBUGCHK.DMP;
%RMU-F-FTL_REP, Fatal error for REPAIR operation at 29-FEB-2002 09:21:12.44
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2. The RMU/RECLAIM command no longer fails with an access violation error.

3.3.10 RMU/VERIFY/CONSTRAINT Now Uses Warning for CONSTFAIL Message

Enhancement Bug 1644732

In previous releases of Oracle Rdb, RMU/VERIFY/CONSTRAINT would issue an informational message if a constraint failed to verify correctly. This severity was often ignored by log file summarizers and so the severity of the CONSTFAIL message has been changed to a warning as shown in the following example.

```
$ RMU /VERIFY /CONSTRAINT SQL$DATABASE
%RMU-W-CONSTFAIL, Verification of constraint "T_CHECK1" has failed.
%RMU-W-CONSTFAIL, Verification of constraint "T_CHECK2" has failed.
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.11 RMU Prompt to Operator Console Ignored Correct Responses

If prompts were directed to the operator console for RMU/BACKUP and RMU/RESTORE and not to the user's terminal, the length of the response entered by the operator was incorrectly returned, causing the operator to be reprompted even if he entered a valid response.

This problem happened even though the operator's response to the prompt was one of the following valid responses:

- INITIALIZE
- QUIT
- RETRY
- UNLOAD

The following example shows that even though the operator responded to the prompt with a valid response, it was ignored and he was reprompted.

```
$RMU /BACKUP /LOG /REWIND /LABEL=(LABEL1,LABEL2) -
 MF_PERSONNEL.RDB TAPE1:MF_PERSONNEL.BCK, TAPE2:

%MOUNT-I-MOUNTED, LABEL1 mounted on _TAPE1: (HJ50AC)
%RMU-I-WRNGLBL, Tape on _TAPE1 was incorrectly labeled. Expected LABEL1 -
Found XXX
%RMU-I-TAPEDISPW, Specify tape disposition for _TAPE1 (QUIT,INITIALIZE,
RETRY,UNLOAD)
INITIALIZE

%RMU-I-TAPEDISPW, Specify tape disposition for _TAPE1 (QUIT,INITIALIZE,
RETRY,UNLOAD)
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.12 RMU Incremental Backup and Restore Could Cause Truncated Table Rows to Reappear

Bugs 1926428 and 1987848

There was a problem with RMU/BACKUP/INCREMENTAL and RMU/RESTORE/INCREMENTAL where rows deleted by a truncate table command in SQL could reappear following an incremental RMU/RESTORE of uniform storage areas where a truncate table operation had taken place since the last full backup. This

happened because RMU/BACKUP/INCREMENTAL and RMU/RESTORE/INCREMENTAL did not save and restore the status of deleted rows from truncated tables as having been deleted due to a truncate table operation.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.13 Deleted Rows Reappear After RMU/REPAIR

Bug 1926428

If we truncated a table and ran RMU/REPAIR/SPAM twice or RMU/REPAIR/INIT=FREE, the deleted rows reappeared making the database unreliable.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

Note that /SPAM is the default qualifier on the RMU/REPAIR command only if a user does not specify any of the following qualifiers on the RMU/REPAIR command line:

- /AIP
- /ABM
- /INITIALIZE = FREE_PAGES
- /INITIALIZE = SNAPSHOTS
- /INITIALIZE = SNAPSHOTS = CONFIRM

Previously /SPAM was the default qualifier on all RMU/REPAIR commands.

3.3.14 RMU/EXTRACT Incorrectly Extracts Index STORE Clause When Using GROUP_TABLE Option

Bug 2270186

In prior releases of Oracle Rdb 7.1, the GROUP_TABLE option did not correctly extract the STORE clause for indices. The STORE keyword was missing and thus the definition was invalid.

The following example shows this.

```
create index STORE_INDEX1
    on STORE_TAB1 (
    A1
        asc)
    type is SORTED
    in STORE1;
    type is SORTED
    using (A1)
        in STORE1
            with limit of (5)
        otherwise in STORE2;
```

As a workaround, the output from RMU/EXTRACT can be edited to include the missing STORE clause, or the GROUP_TABLE option can be omitted.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.15 RMU/CONVERT/NOCOMMIT to V71 Lock Conflict Within Default Storage Area

Bug 2268086

In most cases in RMU/CONVERT, when the RDB$SYSTEM storage area is readied, the default storage area (if it is different from RDB$SYSTEM) is also readied. This must be done so that system tables contained in logical areas within the default storage area can be accessed. However, there was a case where the RDB$SYSTEM storage area was readied but the default storage area was not also readied. This was the case where RMU/CONVERT/NOCOMMIT was followed by RMU/CONVERT/COMMIT and a default storage area other than RDB$SYSTEM was defined which contained system tables. Therefore, when RMU/CONVERT attempted to access system tables in logical areas within the default storage area, and the default storage area was not readied for access, lock conflicts occurred. This problem will not happen if you do not specify /NOCOMMIT when you do RMU/CONVERT to V71 or if you do not have a default storage area other than RDB$SYSTEM defined for the database being converted.

The following example shows that a lock conflict occurred when an RMU/CONVERT/NOCOMMIT of a database to Rdb V71 was followed by an RMU/CONVERT/COMMIT and a default storage area other than RDB$SYSTEM was defined for the database.

```
$RMU /CONVERT /NOCOMMIT TEST
%RMU-I-RMUTXT_000, Executing RMU for Oracle Rdb X7.1-00
Are you satisfied with your backup of DEVICE:[DIRECTORY]TEST.RDB;1
 and your backup of any associated .aij files [N]? Y
%RMU-I-LOGCONVRT, database root converted to current structure level
RMU-S-CVTDBSUC, database DEVICE:[DIRECTORY]TEST.RDB;1
 successfully converted from version V7.0 to V7.1
$RMU /CONVERT /COMMIT TEST
%RMU-I-RMUTXT_000, Executing RMU for Oracle Rdb X7.1-00
Are you satisfied with your backup of DEVICE:[DIRECTORY]TEST.RDB;1
 and your backup of any associated .aij files [N]? Y
%RMU-I-LOGCONVRT, database root converted to current structure level
%RMU-F-LCKCNFLCT, lock conflict on logical area 23
%RMU-F-FTL_CNV, Fatal error for CONVERT operation at 18-MAR-2002 08:38:14.88
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.16 RMU/COLLECT OPTIMIZER_STATISTICS Fails When Temporary Tables in Database

Bug 2245491

In previous releases of Oracle Rdb, the RMU/COLLECT OPTIMIZER_STATISTICS command would fail if there were temporary tables in the database that also had storage maps defined. The storage maps can be used to disable compression as shown in this example.

```
SQL> create global temporary table GT (a integer);
SQL> create storage map GT_MAP for GT
cont>    disable compression;
```

When this database was processed using RMU/COLLECT the following error would occur:

```
$ RMU /COLLECT OPTIMIZER_STATISTICS TEST_DB/LOG
Start loading tables... at 21−MAR−2002 14:39:50.24
%SYSTEM−F−ACCVIO, access violation, reason mask=04, virtual
address=0000000000000068, PC=0000000000345B14, PS=0000001B
%RMU−F−FATALOSI, Fatal error from the Operating System Interface.
%RMU−I−BUGCHKDMP, generating bugcheck dump file DISK1:[TEST_DB]RMUBUGCHK.DMP;
%RMU−F−FTL_ANA, Fatal error for ANALYZE operation at 21−MAR−2002 14:39:50.85
```

A workaround for this problem is to drop just the storage maps for the temporary tables. RMU/COLLECT normally ignores views and temporary tables. Once the RMU/COLLECT command has been executed, the storage maps can be re−created for the temporary tables.

This problem has been corrected in Oracle Rdb Release 7.1.0.2. RMU/COLLECT now correctly filters temporary tables that also have storage maps.

3.3.17 RMU/BACKUP and RESTORE RMU−I−RESUME Message Gave Incorrect Volume Number

The RMU/BACKUP and RESTORE RMU−I−RESUME message could give an incorrect large volume number at end of volume when switching from one tape volume to another. For example:

```
%RMU−I−RESUME, resuming operation on volume 12445679 using _$111$MUA31
```

instead of

```
%RMU−I−RESUME, resuming operation on volume 2 using _$111$MUA31
```

The internal volume number used by RMU was correct but there was a problem putting out the volume number in the message. This has been corrected.

The following example shows that even though the internal volume number used by RMU was correct an incorrect large volume number was output in the RMU−I−RESUME message at volume change.

```
$RMU/RESTORE/DIR=DEVICE:[DIRECTORY]/NOCDD/LABEL=TEST/REWIND/LOG/VOLUMES=2 −
 $111$MUA31:TEST.RBF
%RMU−I−AIJRSTBEG, restoring after−image journal "state" information
%RMU−I−AIJRSTEND, after−image journal "state" restoration complete
%RMU−I−RESTXT_00, Restored root file DEVICE:[DIRECTORY]TEST.RDB;1
%RMU−I−RESTXT_21, Starting full restore of storage area
 DEVICE:[DIRECTORY]DATA.RDA;1 at 27−MAR−2002 07:55:22.71
%RMU−I−RESUME, resuming operation on volume 12445679 using _$111$MUA31
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.18 RMU/RESTORE Access Violation on Ready Volume Prompt to Operator Console

If RMU/RESTORE was restoring from one or more tape devices and a tape volume was not ready, an access violation occurred just before the prompt was going to be output to the operator console from a batch job. This did not happen if the prompt was output to the user terminal from an interactive RMU/RESTORE command.

Therefore, the access violation occurred and the prompt was never output to the operator console.

The following example shows that an access violation occurred instead of the prompt to the operator console to ready the next volume.

```
$RMU/RESTORE/DIRECTORY=DEVICE:[DIRECTORY]/NOCDD/LABEL=TEST/REWIND/LOG-
/VOLUMES=2 $111$MUA31:TEST.RBF
%RMU-I-AIJRSTBEG, restoring after-image journal "state" information
%RMU-I-AIJRSTEND, after-image journal "state" restoration complete
%RMU-I-RESTXT_00, Restored root file DEVICE:[DIRECTORY]TEST.RDB;1
%RMU-I-RESTXT_21, Starting full restore of storage area
 DEVICE:[DIRECTORY]DATA.RDA;1 at 27-MAR-2002 07:55:22.71
%RMU-I-RESUME, resuming operation on volume 2 using _$111$MUA31
%MOUNT-F-MEDOFL, medium is offline
%SYSTEM-E-ACCVIO, access violation, reason mask=00, virtual
address=0000000000000000, PC=0000000000395568, PS=0000001B
%RMU-I-BUGCHKDMP, generating bugcheck dump file
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.19 RMU/CONVERT to V71 Errors

The following four problems have been discovered when doing RMU/CONVERT to V71.

These problem have been corrected in this release of Oracle Rdb, Release 7.1.0.2. Databases converted with this and future Rdb 7.1 releases will not exhibit these problems. However, databases which were previously converted will contain these errors in conversion.

Fortunately these problems do not affect the running of applications on the affected databases. However, it is possible that the incorrect domain names stored for RDB$PARAMETERS will cause incorrect scripts to be generated by RMU/EXTRACT and incorrect definitions in a SQL EXPORT file.

Oracle Rdb Engineering has created a tool which can be run on Rdb Release 7.1.0 and Rdb Release 7.1.0.1 databases which will repair these problems. This tool can be run online and is available for download on the Oracle MetaLink Patch Download area. Please contact Oracle Support for further information.

3.3.19.1 RMU/CONVERT to V71 Changed the Value of Some Existing System Table Fields

Bug 2245306

RMU/CONVERT to V71 changed the value of some existing system table creator and related date fields instead of preserving the existing values of these fields.

Now the existing system table creator and date fields which were getting modified or initialized by RMU/CONVERT (RDB$FIELD_CREATOR in RDB$FIELDS, RDB$RELATION_CREATOR in RDB$RELATIONS and RDB$MODULE_CREATOR in RDB$MODULES) as well as the related RDB$CREATED and RDB$LAST_ALTERED timestamps will not be modified. The current value as it was before the convert will be preserved.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.19.2 RMU/CONVERT to V71 Truncated the RDB$PARAMETER_SOURCE Value in RDB$PARAMETERS

Bug 2307045

RMU/CONVERT to V71 did not correctly copy the value of the RDB$PARAMETER_SOURCE field from the existing RDB$PARAMETERS system table to the converted V71 RDB$PARAMETERS system table since it ignored a change in field alignment between V70 and V71. The first character of the RDB$PARAMETER_SOURCE field would be missing in V71.

The following example shows that the first character in the RDB$PARAMETER_SOURCE field in the RDB$PARAMETERS system table was missing in the database converted to V71 from V70.

Here is the RDB$PARAMETER_SOURCE value in the V70 database.

```
SQL> select RDB$PARAMETER_SOURCE from RDB$PARAMETERS;
 RDB$PARAMETER_SOURCE
 TEST_DOMAIN
```

Here is the truncated RDB$PARAMETER_SOURCE value in the converted V71 database.

```
SQL> select RDB$PARAMETER_SOURCE from RDB$PARAMETERS;
 RDB$PARAMETER_SOURCE
 EST_DOMAIN
```

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.19.3 RMU/CONVERT to V71 Gave Incorrect Values to Some Fields in RDB$CONSTRAINTS

RMU/CONVERT to V71 did not correctly convert the values of the RDB$CREATED, RDB$LAST_ALTERED, RDB$CONSTRAINT_CREATOR and RDB$SECURITY_CLASS fields in the RDB$CONSTRAINTS system table when converting databases to Oracle Rdb V71. The values were shifted to the right from the correct starting position in the field.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.3.19.4 SHOW SEQUENCE Displays Strange Value for NEXT SEQUENCE VALUE

Bug 2325235

When SHOW SEQUENCE is used, an unexpected value is displayed for the Next Sequence Value attribute. This problem only occurs when a database has been converted to Oracle Rdb V7.1 using RMU/CONVERT or RMU/RESTORE from a prior version. Databases created using CREATE DATABASE or IMPORT DATABASE do not have this problem.

The following output shows an example of this unexpected value.

```
SQL> show sequence ID_SEQUENCE
Sequences in database with filename testdb
```

```
    ID_SEQUENCE
 Sequence Id: 1
 Initial Value: 215585
 Minimum Value: 1
 Maximum Value: 9223372036854775806
 Next Sequence Value: 45213529323200000
 Increment by: 1
 Cache Size: (Disabled)
 No Order
 No Cycle
 No Randomize
```

Queries on the sequence and output from RMU/DUMP/HEADER show that the next sequence value is not such a high value.

```
SQL> select ID_SEQUENCE.nextval from rdb$database;

               221692
1 row selected

$ RMU/DUMP/HEADER TESTDB
   .
   .
   .
Client sequences:
    - 32 client sequences have been allocated
    Sequence #1. is active
      - Current value = 221693.
      - Flags mask = 00000000
      - Reserved Flags = 00000001
    - 1 client sequence in use
```

This problem was caused by an error in RMU/CONVERT which didn't fully describe the special COMPUTED BY column RDB$NEXT_SEQUENCE_VALUE in the table RDB$SEQUENCES. However, the sequence continues to function correctly. Only SHOW SEQUENCE, EXPORT, and RMU/EXTRACT will fetch the wrong value from this column.

This problem has been corrected in Oracle Rdb Release 7.1.0.2. RMU/CONVERT now correctly populates this column. In addition, this release of Oracle Rdb correctly handles this column even for databases converted to V7.1 and V7.1.0.1 by older versions of RMU/CONVERT.

3.4 Row Cache Errors Fixed

3.4.1 Bugchecks in PIOGB$PURGE_BUFFER After Node Failure When Row Cache in Use

Bug 2058891

When the Row Cache feature was enabled with global buffers, it was possible for processes to bugcheck with the following exception after a node failure occurred:

```
***** Exception at 00E58F9C : PIOGB$PURGE_BUFFER + 0000078C
%COSI-F-BUGCHECK, internal consistency failure
```

The problem could also occur the first time the database was accessed after an RMU/CLOSE/ABORT=DELPRC command was issued.

There was a problem in the database recovery mechanisms for the Row Cache feature that could cause global buffer data structures to become inconsistent.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5 RMU Show Statistics Errors Fixed

3.5.1 RMU/SHOW STATISTICS Does Not Honor CHECKPOINT_SORT

Bug 2057091

There was a problem wherein the CHECKPOINT_SORT in the RMU/SHOW STATISTICS configuration file was not being honored.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.2 RMU/SHOW STATISTICS CHECKPOINT_ALARM Does Not Give Out OPCOMs

Bug 1735654

The CHECKPOINT_ALARM variable is no longer used to give out operator notification messages (OPCOM) for long transactions. The variable LONG_TX_SECONDS is now used for this purpose. RMU/SHOW STATISTICS gives out OPCOMs to indicate transactions that exceed the interval specified by the LONG_TX_SECONDS at intervals of 1 minute. The OPCOMs are delivered to the OPCOM classes specified by the NOTIFY variable in the configuration file.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.3 Possible RMU Bugcheck or Failure to Notify Triggering of User Defined Events

The notify or invoke associated with a user defined event in RMU/SHOW STATISTICS may not work or an RMU bugcheck may occur when the user−defined event triggers.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.4 AUTO_RECONNECT Variable Value is not Honored When Imported From a RMU/SHOW STATISTICS Configuration File

Bug 2113645

The AUTO_RECONNECT parameter value was not honored when imported from a RMU/SHOW STATISTICS configuration file.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.5 Some RMU/SHOW STATISTICS Counters Can Be Used To Define Events In Interactive Mode But Not In Batch Mode

Bug 2078940

Some RMU/SHOW STATISTICS counters such as "−prom−deadlocks" can be used to define events in interactive mode but not in batch mode.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.6 Stream ID Format is Different in Different Places

Bug 2093770

The Stream ID display has been made uniform everywhere it appears.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.7 RMU/SHOW STATISTICS Online Analysis Configuration Options Do Not Work Properly

Bug 1893049

RMU/SHOW STATISTICS online analysis configuration options did not use the right percentile for displaying read−write and read−only statistics.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.8 Missing "U" for Utility Jobs in RMU/SHOW STATISTICS Displays

Bug 2110027

A "U" was not displayed for utility jobs in RMU/SHOW STATISTICS displays.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.9 RMU/SHOW STATISTICS Mixes Up Count Labels

Bug 1937577

In the RMU/SHOW STATISTICS utility, the count labels associated with row cache search are mixed up.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.10 Errors in Saved RMU/SHOW STATISTICS Configuration File

Bug 1922670

There are three errors in the saved RMU/SHOW STATISTICS configuration file.

- The RUJ_FILE_SIZE parameter is documented to default to 256 but is saved as 25.6 in the configuration file.
- If you are monitoring more than one node and save the configuration file, the current node name is not correctly saved.
- If you are monitoring more than one node, the CLUSTER_NODES parameter is saved with trailing garbage characters.

These problems have been corrected in Oracle Rdb Release 7.1.0.2.

3.5.11 RMU/SHOW STATISTICS Shows Incorrect Area Sizes

Bug 2151237

The RMU/SHOW STATISTICS display of storage area information shows the initial page count statistic two times. Further, the count displayed is not accurate.

This problem has been corrected in Oracle Rdb Release 7.1.0.2. The accurate page count is now displayed only once.

3.5.12 RMU/SHOW STATISTICS Multi−Page Report File

Bug 2195802

In Oracle Rdb Release 7.1.0.1, the RMU/SHOW STATISTICS utility was enhanced to write all pages of multi−page displays to the report file. Unfortunately, this enhancement introduced a regression where some pages were written to the report file multiple times.

This problem has been corrected in Oracle Rdb Release 7.1.0.2. The RMU/SHOW STATISTICS utility now writes all pages of multi−page displays to the report file and writes single−page displays only once.

3.5.13 RMU/SHOW STATISTICS Triggers Invoked From User Defined Events at Times Other Than the Refresh Intervals

Bug 2158913

RMU/SHOW STATISTICS triggers can invoke from a user defined event at times other than refresh intervals. Moreover, the invoke is triggered more than once for each time the threshold is reached. The same event works fine when a "NOTIFY" is used instead of an "INVOKE".

This problem has been corrected in Oracle Rdb Release 7.1.0.2. At present (up to releases 7.0.6.3 and 7.1.0.1), the RMU/SHOW STATISTICS display is updated when the RMU/SHOW STATISTICS keypad is used apart

from being updated at refresh intervals. As a result of the fix for this problem, RMU/SHOW STATISTICS display will only be updated at refresh intervals.

3.5.14 RMU/SHOW STATISTICS Row Cache Information May Not Display the Information of the Cache Selected

Bugs 2220998 and 2150808

RMU/SHOW STATISTICS may not display the information about the correct cache when you select the "Row Cache Information" option.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.5.15 Inconsistency in the Hot Standby Statistics Screen of RMU/SHOW STATISTICS

Bug 1943101

An inconsistency is observed on the Hot Standby Statistics screen of RMU/SHOW STATISTICS. On the standby side, the master AIJ seems smaller than the standby AIJ.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.6 Hot Standby Errors Fixed

3.6.1 7.1.0.1 Process Hangs During AIJ Switchover

In Oracle Rdb Release 7.1.0.1, it was possible to encounter hang problems when using the Hot Standby feature if user processes on the master database had multiple database attaches. This problem was introduced in Release 7.1.0.1.

If a process was attached to multiple databases and the AIJ Log Server (ALS) process was enabled, it was possible for processes to hang with the stall message "hibernating on AIJ submission". One process usually was hung with the stall message "waiting for RTUPB list (EX)". The only way to resolve the problem was to terminate the process that was hanging with "waiting for RTUPB list (EX)".

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

3.6.2 Could Not Use TCP/IP As Hot Standby Network Transport

If TCP/IP was specified as the network transport for Hot Standby, the AIJ Server (RDMAIJ71) process would often fail. The logfile for the server process would contain the following error message:

```
AIJSERVER shutting down: %COSI-F-BUGCHECK, internal consistency failure
```

No bugcheck dump file would be produced by the server process.

This problem has been corrected in Oracle Rdb Release 7.1.0.2.

Chapter 4
Software Errors Fixed in Oracle Rdb Release 7.1.0.1

This chapter describes software errors that are fixed by Oracle Rdb Release 7.1.0.1.

4.1 Software Errors Fixed That Apply to All Interfaces

4.1.1 Excessive Disk I/O for DROP TABLE and TRUNCATE TABLE

Bug 989292

In prior releases of Oracle Rdb, the DROP TABLE and TRUNCATE TABLE statements performed excessive disk I/O when the table contained LIST OF BYTE VARYING columns. When this data type is present, these operations must read the table to locate the LIST data. In prior releases, a DELETE operation was also performed on the table. While this achieved the delete of the LIST data, it also caused constraints, and possibly triggers, to be executed along with updating indices as each row was deleted.

This problem was corrected in Oracle Rdb7 Release 7.0.4 and was inadvertently left out of the Release Notes. The DROP TABLE and TRUNCATE TABLE statements no longer cause constraints and triggers to be executed for the table and indices are no longer updated when processing the LIST OF BYTE VARYING columns. The result is that I/O required for DROP TABLE and TRUNCATE TABLE is significantly reduced, especially for tables stored in UNIFORM format storage areas.

4.1.2 LIST Storage Map Not Updated Upon ALTER or DROP TABLE

Bug 908343

Database administrators can use CREATE STORAGE MAP to establish special storage area mapping for LIST OF BYTE VARYING columns. The LIST storage map can be used to place all or some of the columns of the table in specified storage areas. However, it has been reported that this storage map is not updated when a DROP TABLE or an ALTER TABLE ... DROP COLUMN is executed.

The LIST data is deleted from the database, however, the name of the table or column is left in the storage map. This leads to confusion later when RMU/EXTRACT is used to process the storage map. Further, if columns from the table were the only data stored in that partition, Rdb would not delete the logical area when the table was dropped.

These problems have been corrected in Oracle Rdb Release 7.1. Oracle Rdb now implicitly updates the LIST storage map when you drop a referenced table or column.

4.1.3 ARBs Exhausted

It was possible for a database to run out of AIJ Request Blocks (ARBs) if many processes were abnormally terminated. If a process had an ARB allocated at the time it was terminated, the Database Recovery Process (DBR) would fail to free the ARB allocated to the process. This problem was introduced in Oracle Rdb Release 7.0.1.2.

Symptoms of this problem include:

- Processes looping. RMU/SHOW STATISTICS would show processes stalling waiting for the AIJ lock or writing the same AIJ block over and over.
- More AIJ activity due to processes flushing the ARBs more often in attempts to make ARBs available.
- The "AIJ Journal Information" screen displayed by RMU/SHOW STATISTICS would show the available ARB count ("ARB.Avail:") to be few or none.

To avoid the problem, avoid terminating processes via the DCL STOP /IDENTIFICATION command. When the problem occurs, the database must be closed and re−opened on each node where the problem is being seen to reset the free ARB lists.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.4 CLEAN BUFFER COUNT Parameter Not Obeyed

When the Asynchronous Batch Write feature is being used, Oracle Rdb is supposed to inspect the tail of the least recently used (LRU) buffer queue to determine if there are any modified buffers at the end of the queue. The CLEAN BUFFER COUNT parameter specifies how many buffers are to be inspected. If any are found then those buffers are supposed to be written to disk. However, when unmarking buffers, Oracle Rdb would unmark buffers at the end of the modified queue instead of the LRU queue. That could cause buffers that were just modified to be immediately written, even if they were the most recently accessed buffers. This could cause the buffer to have to be modified again and thus written again.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. Instead of writing the buffers at the tail of the modified queue, Oracle Rdb now writes the modified buffers at the end of the LRU queue.

4.1.5 DETECTED ASYNCHRONOUS PREFETCH THRESHOLD Not Obeyed

The detected asynchronous prefetch (DAPF) feature is supposed to initiate asynchronous prefetch (APF) requests if it detects consecutive pages being fetched from a storage area. The THRESHOLD parameter declares how many consecutive buffers read in a sequence will trigger an APF request. However, Oracle Rdb would not actually initiate APF requests until the THRESHOLD count plus half the DEPTH number of buffers were sequentially read.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. DAPF will now be triggered when THRESHOLD number of consecutive buffers are read in a sequence.

4.1.6 Page Locks Not Demoted at End of Transaction When FAST COMMIT Enabled

When using the FAST COMMIT feature, at the end of a transaction, page locks were not being demoted. Page locks are always demoted at the end of a transaction when the FAST COMMIT feature is not enabled. In some applications, demoting page locks at the end of a transaction can significantly reduce the incidence of deadlocks involving page locks.

This situation has been improved in Oracle Rdb Release 7.1.0.1. When the FAST COMMIT feature is enabled, at the end of a transaction, any buffer that does not contain a modified page will have its page locks demoted.

4.1.7 Bitmapped Scan Causes Bugcheck on Transaction Termination

Bug 1978724

A problem with the way bitmapped scan uses indexes in the dynamic optimizer to carry out the scan caused bugchecks on transaction or session termination.

The call stacks of these bugcheck dumps may include the following:

```
KOD$ROLLBACK + 00000154
%COSI−F−BUGCHECK, internal consistency failure

or

KOD$PREPARE + 00000288
```

This problem may occur when the dynamic optimizer determines that a query may be satisfied by three or more indexes, the first priority index chosen being a non−ranked index (that is, either a normal sorted or a hashed index). At least two of the remaining indexes have to be sorted ranked indexes for the optimizer to choose to implement the 'bitmapped scan' optimization.

An example of the portion of the strategy dump from a query that will exhibit this behavior follows:

```
Leaf#01 FFirst CLIENT_DATA Card=5001    Bitmapped scan
  BgrNdx1 HASHED_1 [(1:1)2] Fan=1
  BgrNdx2 RANKED_3 [1:1] Fan=82
  BgrNdx3 RANKED_2 [1:1] Fan=82
  BgrNdx4 NON_RANKED_1 [1:1] Fan=82
```

A possible workaround for this problem is to disable bitmapped scans by either:

```
set flags 'nobitmapped_scan;

or

$ define RDMS$DISABLE_BITMAPPED_SCAN "1"
```

Disabling bitmapped scan optimization does not stop bitmapped indexes from being used for data retrieval.

Another possible workaround is to either change the first index chosen by the dynamic optimizer to a ranked index or to disable that index entirely.

This problem does not cause any data corruption in your database.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.8 Problems With Column Outlines

Two problems have been found with the creation of outlines on COMPUTED BY columns.

1. Bugcheck dumps may be seen when trying to create outlines on COMPUTED BY columns that use aggregate functions such as MAX or MIN.
   For example, attempting to create an outline on the following COMPUTED BY column would generate a bugcheck dump.

   ```
   F1 computed by (select MAX(job_end) from JOB_HISTORY)
   ```

   There is no known workaround for this problem.
2. If two or more COMPUTED BY columns exist on the same table, and at least one of these columns has an outline created on it, it is possible that when the optimizer tries to optimize a query using these outlines, the query optimization will fail and the query will be aborted with the following error message:

   ```
   %RDMS-F-LEVEL_MISMATCH, the table/subquery nesting levels in the query outline
   do not match the query
   ```

   This problem may occur when a query references at least two COMPUTED BY columns from the same table and one of these has an outline stored for it.
   Possible workarounds for this problem are to drop the offending outline or to disable outlines by using the SET FLAGS 'IGNORE_OUTLINES' statement.

These problems have been corrected in Oracle Rdb Release 7.1.0.1.

4.1.9 Count Scan Optimization Incorrectly Returning Count of 0

Bug 2020109

A problem in the new COUNT SCAN optimization used with ranked indexes may cause incorrect results to be returned by COUNT. Depending on the distribution of keys within the ranked index nodes and the search criteria provided to the COUNT statement, the COUNT statement may incorrectly return a value of 0.

This problem will only occur when the optimizer uses count scan optimization on a sorted ranked index where the search criteria provided in the selection expression for the COUNT statement generates a search key that does not match an existing key within the index. Depending on key distribution, the scan may, infrequently, terminate prematurely resulting in an incorrect value of 0 being returned.

A possible workaround for this problem is to disable count scan optimization by using the SET FLAGS statement or logical name, as in the following example.

```
SQL> SET FLAGS 'NOCOUNT_SCAN';

or

$ DEFINE RDMS$SET_FLAGS 'NOCOUNT_SCAN'
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.10 Disabling AIJ When Row Cache Recovery Required

Bug 1831040

When after−image journaling is manually disabled on a closed database that had Row Caching active and requires recovery, it is possible to render the database unusable. For example, consider the following sequence of events:

1. Database is running with Row Caching enabled
2. AIJ files not backed up and eventually fill
3. User processes deleted or system fails
4. User enters RMU /SET AFTER_JOURNAL /DISABLE command

At this point, a warning message is displayed, but the database can not be opened because the DBR process will fail when attempting to access the after image journal files.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. Attempts to disable journaling will now result in a fatal error and journaling will not be disabled when Row Cache recovery is required. The following example demonstrates this condition.

```
$ RMU /SET AFTER /DISABLE MF_PERSONNEL.RDB
%RMU−W−DBRABORTED, database recovery process terminated abnormally
%RMU−F−MUSTRECDB, database must be closed or recovered
%RMU−F−FTL_SET, Fatal error for SET operation at 11−SEP−2001 22:52:22.37
```

4.1.11 Bitmapped Scan Problem With Large Indexes

Bug 2030599

A problem in the new bitmapped scan optimization used with ranked indexes may infrequently cause Rdb to return zero records even when matching records exist.

This problem may be found only when either the data records associated with the keys stored in the ranked indexes span more than 131070 pages or if the data records span over 3 or more areas. In addition, the existence of this problem depends strongly on the distribution of those records and the selection criteria used to match records across the indexes.

Bitmap scan optimization may be chosen by the optimizer when two or more ranked indexes are found that may satisfy all or part of the selection criteria of a query.

Dumping the query strategy using the 'STRATEGY' debug flag will show those queries that have been optimized this way. At the end of the LEAF information of the strategy dump will be the phrase 'Bitmapped scan', as in the following example.

```
Leaf#01 FFirst CUSTOMER_DATA Card=5065237 Bitmapped scan
  BgrNdx1 ADDR_INDEX [1:1] Fan=82     (index scan#2)
  BgrNdx2 NAME_INDEX [1:1] Fan=82     (index scan#3)
  BgrNdx3 POSTCODE_INDEX [1:1] Fan=82 (index scan#4)
```

A possible workaround for this problem is to disable bitmapped scan optimization by using the SET FLAGS statement or logical name.

For example:

```
SQL> SET FLAGS 'NOBITMAPPED_SCAN';

or

$ DEFINE RDMS$SET_FLAGS 'NOBITMAPPED_SCAN'
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.12 Query With Range List OR Predicates Returns Wrong Results

Bug 1329838

The following query with range list OR predicates returns wrong results.

```
set flags 'strategy,detail';

select t,m,p,b from a
  where (t='S' and (m='N' or p='Q')) or (t='Z' and (m='N' or b='A'))
  order by t,m,p,b;
Tables:
  0 = A
Sort: 0.T(a), 0.M(a), 0.P(a), 0.B(a)
Conjunct: ((0.T = 'S') AND ((0.M = 'N') OR (0.P = 'Q'))) OR ((0.T = 'Z') AND ((
          0.M = 'N') OR (0.B = 'A')))
OR index retrieval                          ! <== Let's call this "Outer"
  Conjunct: (0.B = 'A') OR (0.M = 'N') OR (0.M = 'N')
  OR index retrieval                        ! <== let's call this "Inner"
    Get     Retrieval by index of relation 0:A
      Index name  BTY_X [2:2]
        Keys: (0.B = 'A') AND (0.T = 'Z')
    Conjunct: NOT (0.B = 'A') AND ((0.M = 'N') OR (0.M = 'N')) ! <== incorrect
    Get     Retrieval by index of relation 0:A
      Index name  MTZ_X [(2:2)2]
        Keys: r0: (0.M = 'N') AND (0.T = 'S')
              r1: (0.M = 'N') AND (0.T = 'Z')
  Conjunct: NOT ((0.B = 'A') OR (0.M = 'N') OR (0.M = 'N')) ! <== incorrect
  Get     Retrieval by index of relation 0:A
    Index name  PZY_X [1:1]
      Keys: 0.P = 'Q'
 T       M           P         B
 S       M           Q         B
 S       M           Q         NULL
 S       N           P         B
 S       N           P         NULL
 S       N           Q         B
 S       N           Q         NULL
 S       N           NULL      B
 S       N           NULL      NULL
 S       NULL        Q         B
 S       NULL        Q         NULL
```

```
10 rows selected

where the sequential access gives the correct result:

select t,m,p,b from a
  where (t='S' and (m='N' or p='Q')) or (t='Z' and (m='N' or b='A'))
  order by t,m,p,b optimize for sequential access;
 T      M           P        B
 S      M           Q        A               <= missing row
 S      M           Q        B
 S      M           Q        NULL
 S      N           P        A               <= missing row
 S      N           P        B
 S      N           P        NULL
 S      N           Q        A               <= missing row
 S      N           Q        B
 S      N           Q        NULL
 S      N           NULL     A               <= missing row
 S      N           NULL     B
 S      N           NULL     NULL
 S      NULL        Q        A               <= missing row
 S      NULL        Q        B
 S      NULL        Q        NULL
15 rows selected
```

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

1. The main select query contains a where clause with range list OR predicates that involves four columns, each testing equality with a constant literal value. In this example, we use the column names B, M, P, and T.
2. The column T is a common segment between index BTY_X and MTZ_X, where BTY_X is an index on columns B, T and Y; MTZ_X is an index on columns M, T, and Y. The column P is defined as a leading segment in PZY_X.
3. The main OR predicate has the left branch which contains an AND between "T='S'" and another secondary OR predicate "(m='N' or p='Q')". The right branch contains an AND between "T='Z'" and another secondary OR predicate "(m='N' or b='A')".
4. The OR predicates are arranged in such a way that the strategy of the optimizer uses the range list retrieval "MTZ_X [(2:2)2]" on keys "r0: (0.M = 'N') AND (0.T = 'S')" and "r1: (0.M = 'N') AND (0.T = 'Z')" in the second leg of the "inner" OR index retrieval under the first leg of the "outer" OR index retrieval.
5. The NOT filter, created at the top of the second leg of the "inner" OR index retrieval, does not contain the equality predicate "0.T = 'Z'" from the first leg.
6. The NOT filter, created at the top of the second leg of the "outer" OR index retrieval, does not contain the predicates "(0.T = 'S')" and "(0.T = 'Z')" from the range list predicates of the first leg.

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.13 Database Corruption Using Cluster With Galaxy and Non−Galaxy Nodes

It was possible for page updates to be lost when the following conditions were true:

- The database had GALAXY SUPPORT IS ENABLED.
- The database had GLOBAL BUFFERS ENABLED.
- The database was being accessed concurrently by both OpenVMS Galaxy and non−Galaxy nodes.
- The database was often being closed and reopened on one or more of the Galaxy nodes, but never closed on all of the Galaxy nodes at the same time.

In the above situation, it was possible for updates made by a non−Galaxy node to be lost if the non−Galaxy node closed the database and pages modified by the non−Galaxy node were also present in the global buffer pool being shared by the Galaxy nodes, and those pages in the Galaxy global buffer pool were not being used by any of the Galaxy nodes at the time the database was closed by the non−Galaxy node.

Any of the following actions can be taken to workaround the problem:

- Disable GALAXY SUPPORT.
- Disable GLOBAL BUFFERS.
- Manually open the database on all Galaxy nodes and keep the database open on all Galaxy nodes until all users accessing the database from the Galaxy nodes detach from the database.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.14 Performance Problems when RDM$BIND_SNAP_QUIET_POINT Defined to 0

Bug 884004

When the logical name RDM$BIND_SNAP_QUIET_POINT was defined to 0, it would cause Oracle Rdb to write out modified buffers and demote all page buffer locks when a READ ONLY transaction was started. This would defeat the optimizations utilized by the FAST COMMIT feature and would also cause additional locking and page buffer I/O.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. When the RDM$BIND_SNAP_QUIET_POINT logical is defined to 0 and a process is holding the quiet point lock when starting a READ ONLY transaction, the quiet point lock will be retained. Thus buffers will not be flushed and page locks will not be released when starting a READ ONLY transaction. If a backup process requests the quiet point lock, and the logical RDM$BIND_SNAP_QUIET_POINT is defined to 0, then any READ ONLY transactions will immediately write out modified buffers and release the quiet point lock.

4.1.15 Workload Ignored When Loaded with RMU/INSERT OPTIMIZER_STATISTICS

In previous versions of Oracle Rdb, if workload statistics were loaded into a database using the *RMU/INSERT OPTIMIZER_STATISTICS* command, the workload would be ignored by the optimizer.

The use of workload statistics can be observed by setting the *ESTIMATES* debug flag as shown in the following example.

```
SQL> set flags 'estimates';
SQL> select * from t1 where f1=1;
Solutions tried 1
Solutions blocks created 1
```

```
Created solutions pruned 0
Cost of the chosen solution   3.0000000E+00
Cardinality of chosen solution   1.0000000E+00
~O: Workload statistics used
          F1            F2
           1             1
1 row selected
```

After loading workload statistics with the *RMU/INSERT* command, a query that should use statistics will fail to show the *~O: Workload statistics used* message. This indicates that the statistics are being ignored.

The problem can be identified by examining the data loaded into the *RDB$WORKLOAD* system table. If the *RDB$CREATED* and *RDB$LAST_ALTERED* columns have the same value, as shown in the following example, then workload statistics will be ignored.

```
SQL> select rdb$created,rdb$last_altered from rdb$workload;
 RDB$CREATED               RDB$LAST_ALTERED
 19-OCT-2001 00:33:53.27   19-OCT-2001 00:33:53.27
1 row selected
```

The problem can be corrected by manually updating the *RDB$LAST_ALTERED* column, as shown in the following example. New attaches will commence using the workload values.

```
SQL> update rdb$workload set rdb$last_altered=current_timestamp
cont>where rdb$relation_name='...';
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.16 Descending Sort Not Producing Correct Ordering for BIGINT and DATE Columns

Bugs 2064232 and 2058531

Oracle Rdb Release 7.1 introduced a new fast sort facility (known as QSORT) which is used when the number of rows to be sorted are few and the sort keys are simple.

Unfortunately, QSORT did not correctly handle descending sorts for 64 bit values, such as BIGINT, DATE (both VMS and ANSI formats), TIME, TIMESTAMP and INTERVAL.

A workaround for this problem is to disable QSORT and revert to the normal sort interface by defining the following logical name to the value zero (0).

```
$ DEFINE RDMS$BIND_MAX_QSORT_COUNT 0
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.17 Bitmapped Scan Incorrectly Chosen by Optimizer

A problem in the way the Rdb optimizer determines when to use the new bitmapped scan optimization used with ranked indexes may infrequently cause Rdb to return wrong results.

The optimizer may sometimes incorrectly choose to carry out bitmapped scans that are not appropriate given the selection criteria of the query in relation to the columns available to be used within the ranked index columns. See the following example:

```
SQL> att 'file personnel';
SQL> CREATE TABLE bmtest(A INTEGER,B INTEGER,C INTEGER,D INTEGER,E INTEGER);
SQL> INSERT INTO bmtest VALUES(1,1,10,100,1000);
1 row inserted
SQL> INSERT INTO bmtest VALUES(2,1,10,100,1000);
1 row inserted

SQL> SET FLAGS 'STRATEGY';
SQL> SEL * FROM bmtest WHERE B=1;
Conjunct        Get     Retrieval sequentially of relation BMTEST
           A             B             C             D             E
           1             1            10           100          1000
           2             1            10           100          1000
2 rows selected
SQL> SEL * FROM bmtest WHERE B=1 AND D=100;
Conjunct        Get     Retrieval sequentially of relation BMTEST
           A             B             C             D             E
           1             1            10           100          1000
           2             1            10           100          1000
2 rows selected

SQL> SET FLAGS 'NOSTRATEGY';

SQL> CREATE INDEX bmtest_BCA ON bmtest(B,C,A) TYPE IS SORTED RANKED;
SQL> CREATE INDEX bmtest_DEA ON bmtest(D,E,A) TYPE IS SORTED RANKED;

SQL> SET FLAGS 'STRATEGY';
SQL> SEL * FROM bmtest WHERE B=1;
Leaf#01 FFirst BMTEST Card=0
  BgrNdx1 BMTEST_BCA [1:1] Fan=12
           A             B             C             D             E
           1             1            10           100          1000
           2             1            10           100          1000
2 rows selected
SQL> SEL * FROM bmtest WHERE B=1 AND D=100;
Leaf#01 FFirst BMTEST Card=0    Bitmapped scan
  BgrNdx1 BMTEST_BCA [1:1] Fan=12
  BgrNdx2 BMTEST_DEA [1:1] Fan=12
           A             B             C             D             E
           1             1            10           100          1000
1 row selected
```

The last query shows that bitmapped scan has been used but returns incorrect results. Bitmapped scan should not be invoked unless the query provides equality checks for all the columns in the ranked index.

A possible workaround for this problem is to disable bitmapped scan optimization by using the SET FLAGS statement or logical name. See the following example:

```
SQL> SET FLAGS 'NOBITMAPPED_SCAN';

or

$ DEFINE RDMS$SET_FLAGS 'NOBITMAPPED_SCAN'
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.18 Cannot Connect With Remote Access When Using a Logical

Bug 451582

If a logical is used to specify the path in a remote attach, an Rdb 7.1 client fails to connect to the remote database. Depending on the way the database name is specified, either a −RDB−E−BAD_DB_FORMAT or −RDB−F−NONODE is returned. This problem is similar to Bug 451582. The following example shows the problem behavior and the workarounds.

```
ALPHA4> DEFINE LL MALIBU::DISK$USERS:[REMOTE_ACCOUNT]
ALPHA4> SQL
SQL> attach 'filename ll:v70db';
%SQL−F−ERRATTDEC, Error attaching to database ll:my_db
−RDB−E−BAD_DB_FORMAT, ll:v70db does not reference a database known to Rdb
−RMS−E−FNF, file not found
SQL> attach 'filename ll:v70db.rdb';
%SQL−F−ERRATTDEC, Error attaching to database ll:my_db.rdb
−RDB−F−SYS_REQUEST, error from system services request
−RDMS−F−BADDBNAME, can't find database root ALPHA4::DISK$USERS:[REMOTE_ACCOUNT]`
−RDMS−F−NONODE, no node name is allowed in the file specification
SQL> attach 'filename malibu::disk$users:[remote_account]my_db.rdb';
SQL> exit;
ALPHA4> DEFINE LL MALIBU::DISK$USERS:[REMOTE_ACCOUNT]MY_DB.RDB
ALPHA4> SQL
SQL> attach 'filename ll';
SQL>
```

As a workaround, either don't use the logical to specify the path or include the database name in the logical.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.19 Query Joining Derived Tables of Union Legs With Empty Tables Returns Wrong Results

Bug 1818374

The following query, joining two derived tables containing union legs with empty tables, returns wrong results of 0 rows, instead of 1 row.

```
set flags 'strategy,detail';
select c1
  from (select v1.c1 from
        t_02,
        (select * from t_01
            union all
         select * from t_02
         ) v1
         inner join
        (select * from tt_01
            union all
         select * from tt_02
```

```
         ) as v2
        on (v1.c1 = v2.c1 and v1.c2 = v2.c2)) as tmp
  where tmp.c1 = 110759;
Tables:
  0 = T_02
  1 = T_01
  2 = T_02
  3 = TT_01
  4 = TT_02
Merge of 1 entries
  Merge block entry 1
  Cross block of 3 entries
    Cross block entry 1
      Index only retrieval of relation 0:T_02
        Index name  T_02_NDX [0:0]
    Cross block entry 2
      Merge of 1 entries
        Merge block entry 1
        Merge of 2 entries
          Merge block entry 1
          Conjunct: 1.C1 = 110759
          Index only retrieval of relation 1:T_01
            Index name  T_01_NDX [1:1]
              Keys: <mapped field> = 110759
          Merge block entry 2
          Leaf#01 FFirst 2:T_02 Card=1
            Bool: 2.C1 = 110759
            BgrNdx1 T_02_NDX [1:1] Fan=17
              Keys: <mapped field> = 110759
    Cross block entry 3
      Conjunct: 1.C1 = 110759
      Merge of 1 entries
        Merge block entry 1
        Merge of 2 entries
          Merge block entry 1
          Conjunct: (<mapped field> = 3.C1) AND (<mapped field> = 3.C2)
          Index only retrieval of relation 3:TT_01
            Index name  TT_01_NDX [2:2]
              Keys: (<mapped field> = <mapped field>) AND (<mapped field> =
                    <mapped field>)
          Merge block entry 2
          Conjunct: (<mapped field> = 4.C1) AND (<mapped field> = 4.C2)
          Index only retrieval of relation 4:TT_02
            Index name  TT_02_NDX [2:2]
              Keys: (<mapped field> = <mapped field>) AND (<mapped field> =
                    <mapped field>)
0 rows selected

where the tables are defined as :

! table t_01 is empty
create table t_01 (C1   INTEGER);
create index t_01_ndx  on t_01 (C1) ;

! table t_02 has 1 row
create table t_02 (C1  INTEGER, C2   TINYINT);
create index t_02_ndx  on t_02 (C1) ;

insert into t_02  values (110759,9);

! table tt_01 is empty
```

```
create table tt_01 (C1  INTEGER, C2   TINYINT);
create index tt_01_ndx on tt_01 (C1,  C2);

! table tt_02 has 2 rows
create table tt_02 (C1  INTEGER, C2   TINYINT);
create index tt_02_ndx on tt_02 (C1,  C2);

insert into tt_02 values (110759,4);
insert into tt_02 values (110759,9);
```

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

1. The main query selects the column of a derived table with an equality predicate.
2. The main derived table joins a non−empty table (t_02) and an inner join.
3. The inner join involves a derived table of union between an empty table (t_01) and a non−empty table (t_02), and another derived table of union between an empty table (tt_01) and a non−empty table (tt_02).

As a workaround, the query works if the empty tables are loaded with some data as in the following example.

```
insert into t_01 values (110759);

select c1
  from (select v1.c1 from
        t_02,
        (select * from t_01
            union all
         select * from t_02
         ) v1
         inner join
        (select * from tt_01
            union all
         select * from tt_02
         ) as v2
        on (v1.c1 = v2.c1 and v1.c2 = v2.c2)) as tmp
  where tmp.c1 = 110759;
          C1
      110759
1 row selected
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.20 Left Outer Join Query With OR Predicate Returns Wrong Results

Bug 1837522

The following left outer join query with an OR predicate, having an equality predicate of a column and a constant value on the left side, and an equality predicate of a column and a subquery on the right side, returns wrong results. It should find 3 rows, but it only finds 2 rows.

```
set flags 'strategy,detail';
sel job_code, job_start, c1.employee_id, c2.employee_id
  from
```

```
  job_history as c1
  left outer join
  employees as c2 on (c1.employee_id = c2.employee_id)
  where
      c1.job_code = 'JNTR' or
      c1.job_start =
      (select max(job_start) from job_history as c3)
      ;
Tables:
  0 = JOB_HISTORY
  1 = EMPLOYEES
  2 = JOB_HISTORY
Cross block of 2 entries
  Cross block entry 1
    Aggregate: 0:MAX (2.JOB_START)
    Get     Retrieval by index of relation 2:JOB_HISTORY
      Index name  JH_EMPLOYEE_ID [0:0]
  Cross block entry 2
    Conjunct: 0.JOB_START = <agg0>
    Conjunct: 0.JOB_START = <agg0>
    Match    (Left Outer Join)
      Outer loop
        Conjunct: (0.JOB_CODE = 'JNTR') OR (0.JOB_START = <agg0>
        Get     Retrieval by index of relation 0:JOB_HISTORY
          Index name  JH_EMPLOYEE_ID [0:0]
      Inner loop      (zig-zag)
        Index only retrieval of relation 1:EMPLOYEES
          Index name  EMP_EMPLOYEE_ID [0:0]
 C1.JOB_CODE   C1.JOB_START   C1.EMPLOYEE_ID   C2.EMPLOYEE_ID
 PRSD           3-Jan-1983    00225            00225
 DMGR           3-Jan-1983    00241            00241
2 rows selected
```

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

1. The main query is a left outer join between 2 tables with an ON clause.
2. The WHERE clause contains an OR predicate, with the left side branch being a simple equality predicate on a column, and the right branch using a sub-query in the equality predicate.

As a workaround, the query works if the left and right side of the OR predicate is swapped. For example:

```
sel job_code, job_start, c1.employee_id, c2.employee_id
  from
  job_history as c1
  left outer join
  employees as c2
  on (c1.employee_id = c2.employee_id)
      where
      c1.job_start =
      (select max(job_start) from job_history as c3)
      or
      c1.job_code = 'JNTR'
      ;
 C1.JOB_CODE   C1.JOB_START   C1.EMPLOYEE_ID   C2.EMPLOYEE_ID
 JNTR           2-Jan-1977    00223            00223
 PRSD           3-Jan-1983    00225            00225
 DMGR           3-Jan-1983    00241            00241
3 rows selected
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.21 Query Using Match Strategy With DISTINCT Function Returns Wrong Results

Bugs 1891938 and 1894192

A query using the match strategy with the Distinct Function returns the wrong results, as in the following example.

```
set flags 'strategy,detail';
select count(*) from
( select distinct
                  t1.ACCOUNT_ID,
                  t1.SECURITY_ID
         from     T1 t1,
                  T2 t2
         where    t1.SECURITY_ID = t2.SECURITY_ID
) as t ;
Tables:
  0 = T1
  1 = T2
Merge of 1 entries
  Merge block entry 1
  Reduce: 0.SECURITY_ID, 0.ACCOUNT_ID
  Sort: 0.SECURITY_ID(a), 0.ACCOUNT_ID(a)
  Conjunct: 0.SECURITY_ID = 1.SECURITY_ID
  Match
    Outer loop
      Sort: 1.SECURITY_ID(a)
      Get Retrieval sequentially of relation 1:T2
    Inner loop (zig-zag)
      Index only retrieval of relation 0:T1
        Index name  T1_NDX1 [0:0]
 ACCOUNT_ID   SECURITY_ID
 A1           DE0005557508
1 row selected

where the tables are defined as :

create table T1 (
    ACCOUNT_ID       CHAR (2),
    SECURITY_ID      CHAR (12) );
create index T1_NDX    on T1 (ACCOUNT_ID, SECURITY_ID);

create table T2 (SECURITY_ID      CHAR (12) );

with the following contents:

select SECURITY_ID from T2;

SECURITY_ID
 DE0005128003
 DE0005557508
2 rows selected

select ACCOUNT_ID,SECURITY_ID from T1;
ACCOUNT_ID   SECURITY_ID
```

```
 A1          DE0005557508
 PP          DE0005128003
2 rows selected
```

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

1. The main query selects from a derived table.
2. The derived table is the output of a distinct query from T1 and T2 with a join column predicate.
3. The join column of table T1 is the second segment in index T1_NDX which is ordered by the first segment ACCOUNT_ID.
4. The order of the join column of table T2 is ascending and different from that of T2.

As a workaround, the query works if the query outline is used to apply cross strategy instead of match, as in the following example.

```
select * from
( select
         distinct
                  t1.ACCOUNT_ID,
                  t1.SECURITY_ID
         from     T1 t1,
                  T2 t2
         where    t1.SECURITY_ID = t2.SECURITY_ID
) as t ;
~S: Outline "QO_325EFDCDDEBFFFA8_00000000" used
Tables:
  0 = T1
  1 = T2
Merge of 1 entries
  Merge block entry 1
  Reduce: 0.ACCOUNT_ID, 0.SECURITY_ID
  Sort: 0.ACCOUNT_ID(a), 0.SECURITY_ID(a)
  Cross block of 2 entries
    Cross block entry 1
      Get Retrieval sequentially of relation 1:T2
    Cross block entry 2
      Conjunct: 0.SECURITY_ID = 1.SECURITY_ID
      Index only retrieval of relation 0:T1
        Index name  T1_NDX [0:0]
-- Rdb Generated Outline : 31-JUL-2001 11:23
create outline QO_325EFDCDDEBFFFA8_00000000
id '325EFDCDDEBFFFA85200828890C4E5BA'
mode 0
as (
  query (
-- For loop
    subquery (
      subquery (
        T2 1    access path sequential
          join by cross to              -- <=== change from match to cross
        T1 0    access path index       T1_NDX
        )
      )
    )
  )
compliance optional     ;
 ACCOUNT_ID   SECURITY_ID
 A1           DE0005557508
 PP           DE0005128003
```

```
2 rows selected
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.22 GROUP BY Query With SUM Aggregate Returns Wrong Results

Bug 1844260

The following GROUP BY query with SUM aggregate returns wrong results (the 1st row of column ESTADO should be 'A' instead of 'V').

```
set flags 'strategy,detail';
select estado, sum(total_dep) from bug_view group by estado;
Tables:
  0 = T1
  1 = T2
Aggregate: 0:SUM (CASE (WHEN (0.ID_PRODUCTO = 20) THEN 20 ELSE 15))
Sort: CASE (WHEN (1.FEC_EXPIRACION > 20001231) THEN 'A' WHEN (((0.ID_PRODUCTO =
      15) OR (0.ID_PRODUCTO = 20)) AND (1.FEC_EXPIRACION <= 20001231)) THEN 'V'
       ELSE NULL)(a)
Conjunct: 0.ID_PRODUCTO = 1.ID_PRODUCTO
Match
  Outer loop      (zig-zag)
    Index only retrieval of relation 0:T1
      Index name  T1_NDX [0:0]
  Inner loop      (zig-zag)
    Get     Retrieval by index of relation 1:T2
      Index name  T2_NDX [0:0]
 ESTADO
 V                          15          <=== ESTADO should be 'A'
 V                          15
2 rows selected

where the view is defined as :

create view bug_view ( id_producto, total_dep, estado ) as
select
         a.id_producto,
         case
             when a.id_producto = 20  then 20
             else 15
             end as total_dep,
         case
             when b.fec_expiracion > 20001231 then  'A'
             when (a.id_producto = 15
                   OR a.id_producto = 20
                   ) and
                  b.fec_expiracion <= 20001231
             then  'V'
             end  as estado
     from opas_saldos_err a, ope_pasiva_err b
     where
              a.id_producto = b.id_producto ;

with the following content in the tables:

select * From t1;
```

```
 ID_PRODUCTO
           8
1 row selected

select * From t2;
 ID_PRODUCTO   FEC_EXPIRACION
           8         20000801
           8         20010628
2 rows selected
```

As a workaround, the query works if the predicate "OR a.id_producto = 20" is commented out from the view, as in the following example.

```
create view bug_view_good ( id_producto, total_dep, estado ) as
select
         a.id_producto,
         case
             when a.id_producto = 20  then 20
             else 15
             end as total_dep,
         case
             when b.fec_expiracion > 20001231 then  'A'
             when (a.id_producto = 15
!                  OR a.id_producto = 20
                   ) and
                  b.fec_expiracion <= 20001231
             then  'V'
             end  as estado
     from t1 a, t2 b
     where
              a.id_producto = b.id_producto ;

select estado, sum(total_dep) from bug_view_good group by estado;
Tables:
  0 = T1
  1 = T2
Aggregate: 0:SUM (CASE (WHEN (0.ID_PRODUCTO = 20) THEN 20 ELSE 15))
Sort: CASE (WHEN (1.FEC_EXPIRACION > 20001231) THEN 'A' WHEN ((0.ID_PRODUCTO =
      15) AND (1.FEC_EXPIRACION <= 20001231)) THEN 'V' ELSE NULL)(a)
Conjunct: 0.ID_PRODUCTO = 1.ID_PRODUCTO
Match
  Outer loop      (zig-zag)
    Index only retrieval of relation 0:T1
      Index name  T1_NDX [0:0]
  Inner loop      (zig-zag)
    Get     Retrieval by index of relation 1:T2
      Index name  T2_NDX [0:0]
 ESTADO
 A                          15
 V                          15
2 rows selected
```

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

1. The main query contains a GROUP BY clause and SUM aggregate function.
2. The SUM aggregate function is defined in the view as a CASE expression.
3. The column in the GROUP BY clause is defined in the view as a CASE expression which contains the same predicate from the CASE expression of the SUM aggregate.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.23 ROLLBACK Hangs Under DECdtm When Called From an ACMS CANCEL Procedure

Bug 1905068

Under certain situations, the CANCEL procedure in an ACMS application would cause the ACMS server process to hang in the RDB dispatch layer. This problem can only occur under the following circumstances:

1. The ACMS application is using 2 phase commit under DECdtm either explicitly (i.e. with a SYS$START_TRAN call) or implicitly (by attaching to multiple Rdb databases).
2. The CANCEL procedure contains a SYS$ABORT_TRAN call or ROLLBACK statement.
3. The ACMS server process has an outstanding pending request which is blocked (e.g. waiting for rows locked by another user).

If all three of these conditions occurred, the ACMS server process would hang in the CANCEL procedure even after the condition that caused the original blocking cleared.

The only workaround is to stop the ACMS server process.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.1.24 COMPUTED BY Columns Now Automatically Reserve Referenced Tables

Bug 1253235

In previous versions of Rdb, it was possible that an application could fail if a reference to a COMPUTED BY or view column required a table not specified in the RESERVING clause of the SET or DECLARE TRANSACTION statement.

The application developer may not know that a column requires these extra tables as part of the transaction, or the definition of the view or COMPUTED BY column may be changed to reference different tables after the application is in production.

The following example shows an example where a COMPUTED BY column (PRICE) requires access to a table (CASE_TABLE) that was not referenced by the RESERVING clause.

```
SQL> set transaction read only
cont>     reserving REPORT_VIEW for shared read;
SQL> select * from REPORT_VIEW order by LINE_NUM;
%RDB-E-UNRES_REL, relation CASE_TABLE in specified request is not a
relation reserved in specified transaction
SQL> rollback;
SQL> set transaction read only
cont>     reserving REPORT_VIEW, CASE_TABLE for shared read;
SQL> select * from REPORT_VIEW order by LINE_NUM;
 CASE_NUM             LINE_NUM                    PRICE
 1                           1                  7270.00
 1                           2                 14540.00
```

```
2 rows selected
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1. Rdb now automatically reserves tables referenced by COMPUTED BY columns for SHARED READ.

4.2 SQL Errors Fixed

4.2.1 Command Line Recall Expanded to 255 Lines

In prior releases of Oracle Rdb, the command line recall was limited to the last 20 lines. This limit has been lifted to 255 (the maximum supported by OpenVMS) for this release of Rdb.

If more recall is required then SQL provides the EDIT command to edit whole statements. This interface currently saves the last 20 commands for edit but the SET EDIT KEEP statement can be used to expand this number.

4.2.2 New Minimum Value for the INTERVAL Leading Precision

In prior releases of Oracle Rdb, the minimum value for the interval leading precision was restricted to two digits. This restriction has been removed: an interval leading precision of 1 is now supported.

The following example shows the support for the lower precision value.

```
SQL> create table TIME_CLOCK
cont>     (employee_id  char(5),
cont>      clock_on             timestamp (2),
cont>      clock_off            timestamp (2),
cont>      shift_duration
cont>          computed by (clock_off − clock_on) hour (1) to minute);
SQL>
SQL> show table (column) TIME_CLOCK
Information for table TIME_CLOCK

Columns for table TIME_CLOCK:
Column Name                    Data Type        Domain
−−−−−−−−−−−                    −−−−−−−−−        −−−−−−
EMPLOYEE_ID                    CHAR(5)
CLOCK_ON                       TIMESTAMP(2)
CLOCK_OFF                      TIMESTAMP(2)
SHIFT_DURATION                 INTERVAL
                                HOUR (1) TO MINUTE
 Computed:       by (clock_off − clock_on) hour (1) to minute
```

As in previous releases, if no precision is provided then a default of 2 digits will be used.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.2.3 Incorrect Processing of CASE Expression

Bug 850442

In prior releases of Oracle Rdb, the SQL interface incorrectly processed CASE expressions which included statistical functions (i.e. COUNT, MAX, MIN, AVG, STDDEV, VARIANCE and SUM).

The following example, which imbeds statistical functions in a CASE expression, caused Rdb to bugcheck:

```
select
    case
        when count(employee_id) >= 1
            then '1'
        when count(employee_id) = 0
            then '2'
        else '3'
    end
 from employees;
%RDMS-I-BUGCHKDMP, generating bugcheck dump file USER2:[TEST]RDSBUGCHK.DMP;
%SQL-I-BUGCHKDMP, generating bugcheck dump file USER2:[TEST]SQLBUGCHK.DMP;
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual
address=0000000000000098, PC=000000000038B948, PS=0000001B
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

This improved handling of statistical functions also corrects some query strategies. The following example implements a simple ABS functionality. Due to the erroneous handling of the statistical function, an extra subselect was present as shown in the optimizer STRATEGY display.

```
SQL> set flags 'strategy';
SQL> select
cont>    case
cont>        when AVG (salary_amount) < 0 then - AVG (salary_amount)
cont>        else AVG (salary_amount)
cont>    end
cont> from SALARY_HISTORY;
Cross block of 2 entries
  Cross block entry 1
    Aggregate       Get     Retrieval sequentially of relation SALARY_HISTORY
  Cross block entry 2
    Aggregate       Get     Retrieval sequentially of relation SALARY_HISTORY

  2.652896707818930E+004
1 row selected
```

The corrected SQL query now only requires a single table access.

```
Aggregate       Get     Retrieval sequentially of relation SALARY_HISTORY

  2.652896707818930E+004
1 row selected
```

Applications that encounter this type of unexpected optimizer strategy will need to be recompiled, and stored procedures and functions will need to be recreated.

4.2.4 ALTER TABLE Not Dropping NOT NULL Constraints When NULL Clause Used

In Oracle Rdb Release 7.1, new syntax was introduced to indicate that a column should allow NULL values. For instance,

```
create table MY_TABLE (my_column integer NULL);
```

This syntax is accepted for compatibility with Oracle RDBMS and on CREATE and ALTER TABLE

prevents the use of the NOT NULL constraint syntax.

When used on ALTER TABLE ... ALTER COLUMN, this clause should also drop any (and all) NOT NULL constraints defined for the column. This was not performed by Rdb Release 7.1.

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

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

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

Constraints referencing table MY_TABLE:
No constraints found

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

Table constraints for MY_TABLE:
No constraints found

Constraints referencing table MY_TABLE:
No constraints found

SQL>
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1. This clause now implicitly drops NOT NULL constraints for the column.

NOTE: Other constraints that prevent NULL values, such as CHECK and PRIMARY KEY, are not affected by this statement. The NULL clause is not a constraint and so is not active beyond the CREATE and ALTER TABLE statements.

4.2.5 Some Constraint Definitions Not Supported for AUTOMATIC Columns

In Oracle Rdb Release 7.1, attempts to define UNIQUE, PRIMARY KEY or FOREIGN KEY constraints for AUTOMATIC columns would result in an error.

In the following example, the programmer desired an automatically generated unique number as a PRIMARY KEY:

```
SQL> create sequence s1;
```

```
SQL> create table t(a automatic as s1.nextval primary key);
%SQL-F-PKCONSNOTCB, Computed column may not be a primary key
```

Only NOT NULL and CHECK constraints were allowed for AUTOMATIC columns.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. All types of constraints are now permitted for AUTOMATIC columns.

4.2.6 %RDB-E-NO_DIST_BATCH_U Error When Executing SET TRANSACTION

Bug 1921672

If a SET TRANSACTION statement was executed to start a distributed transaction (2 phase commit) and which specified certain table partitions, an error was inappropriately returned. Specifically, if partition 14 was named, Rdb would return a %RDB-E-NO_DIST_BATCH_U error and not start the transaction.

For example, suppose an interactive SQL session has two databases attached (this implicitly starts a DECdtm distributed transaction), the following SQL would fail as shown.

```
SQL>SET TRANSACTION READ WRITE WAIT ISOLATION LEVEL READ COMMITTED -
RESERVING DB2.MY_TABLE PARTITION(14) FOR EXCLUSIVE WRITE;
%RDB-E-NO_DIST_BATCH_U, no distributed transaction is allowed with the
recovery mechanism disabled
```

This query will now execute normally and start a distributed transaction.

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.2.7 Select With Identical "not in" Clauses

A SQL query which contained two identical "not in" clauses would cause an application to crash, terminate or bugcheck.

This problem started in Oracle Rdb V7.0.

An example of this type of query follows.

```
select count(*) from JOBS
 where JOB_CODE not in ('A', 'B')
   and JOB_CODE not in ('A', 'B');
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.2.8 Keyword Matching Now Reported by Interactive SQL

In prior versions of Oracle Rdb, the keyword abbreviation and matching support in interactive SQL would discard extraneous characters from a token if an expected keyword matched the leading prefix. This was

confusing in some cases. Interactive SQL now generates an informational message to clearly state the substitution.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

This example shows the informational message generated when extra characters are trimmed from the keyword.

```
SQL> create trigger mytrigger
cont>     after updatete on mytable2
%SQL-I-SPELLCORR, identifier UPDATETE replaced with UPDATE
cont>     (insert into mytable values (mytable2.a, 'Any', 'Value'))
cont>     for each row;
```

4.2.9 CREATE MODULE Bugchecks When a Subselect is Used as a Parameter DEFAULT

In a CREATE MODULE definition, if a subselect was used as a parameter DEFAULT, the create module bugchecked with the following error message:

```
%SQL-F-BUGCHK, There has been a fatal error. Please contact your Oracle support
representative. SQL$BLRXPR - 15
```

An example follows:

```
SQL> create module DEF_MOD
cont>
cont> procedure DEF1
cont>     (in :a integer
cont>      default (select count(*) from rdb$database));
cont> trace :a;
cont>
cont> end module;
%RDMS-I-BUGCHKDMP, generating bugcheck dump file
device:[directory]SQLBUGCHK.DMP;
%SQL-F-BUGCHK, There has been a fatal error. Please contact your Oracle support
representative. SQL$BLRXPR - 15
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1. The CREATE MODULE definition no longer bugchecks.

4.2.10 Obsolete Metadata Errors When Using Rdb SQL V7.1 to Access Oracle Rdb V7.0 Databases

Bug 1994383

When using Oracle Rdb SQL V7.1 to access an Oracle Rdb V7.0 database, obsolete metadata errors were generated when trying to CREATE a TABLE, a VIEW, and/or a DOMAIN.

Specifically, when CREATing a TABLE or a VIEW, the following error message would be generated:

```
CREATE TABLE T (id int);
%RDB-F-OBSOLETE_METADA, request references metadata objects that no longer exist
-RDMS-F-TABNOTDEF, relation RDB$SEQUENCES is not defined in database
CREATE VIEW V as select * from employees;
%RDB-F-OBSOLETE_METADA, request references metadata objects that no longer exist
-RDMS-F-TABNOTDEF, relation RDB$SEQUENCES is not defined in database
```

When trying to CREATE a domain, the following error message would be generated:

```
create domain dom_test int;
%RDB-F-OBSOLETE_METADA, request references metadata objects that no longer exist
-RDMS-F-TABNOTDEF, relation RDB$TYPES is not defined in database
```

These problems have been corrected. SEQUENCES and TYPES are Release 7.1 features and the Rdb SQL code base has been corrected to insure that queries utilizing these features are only performed against V7.1 databases. Thus, error messages are no longer generated.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.2.11 SQL$PRE and SQL$MOD Performance Improvements

Bug 2032243

The performance of the SQL precompiler and the SQL module language compiler has been improved in Oracle Rdb Release 7.1.0.1. This improvement is typically seen as a dramatic reduction in CPU consumption and elapsed time when using the compilers.

Note as well that the size of the SQL$PRE71.EXE and SQL$MOD71.EXE images has been reduced by nearly 50%.

4.2.12 Incompatible Character Sets Not Detected by SQL Interface

In prior versions of Oracle Rdb, the SQL UNION operator would accept incompatible character sets for merging. This incompatibility was only detected at runtime by the Rdb server.

```
SQL> select _dec_mcs'aa' from rdb$database
cont> union
cont> select _kanji'bb' from rdb$database;
%RDB-E-CONVERT_ERROR, invalid or unsupported data conversion
-RDMS-E-CSETBADASSIGN, incompatible character sets prohibit the requested
assignment
```

With this release of Rdb, the SQL interface now detects this error and reports an error indicating the incompatibilities.

```
SQL> select _dec_mcs'aa' from rdb$database
cont> union
cont> select _kanji'bb' from rdb$database;
%SQL-F-INCCSCON, Incompatible character set concatenation between DEC_MCS and
KANJI
```

In addition, SQL now derives a new target character set for the UNION select values by using a character set that is compatible with both. For instance, DEC_KANJI includes the full ASCII character set so it will be chosen as the result character set when ASCII and DEC_KANJI are merged in a UNION operator.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.2.13 SQLMOD Fails to Set Default Character Set Correctly

A problem within SQLMOD prevented the correct default character set from being set for the module compilation if a character set other than DEC_MCS was specified as the DEFAULT CHARACTER SET in the module header.

A check of the listing file will show that the default character set has not been set correctly and, due to this problem, SQL−F−INCCSASS errors may be raised during the module compilation.

For example, the following module tries to set the default character set to SHIFT_JIS, however, the compilation of the module results in compilation errors.

```
$type A.SQLMOD
.
.
.
DECLARE MODULE
    DIALECT SQL92
    DEFAULT CHARACTER SET SHIFT_JIS
    NATIONAL CHARACTER SET SHIFT_JIS
    IDENTIFIER CHARACTER SET SHIFT_JIS
    LITERAL CHARACTER SET SHIFT_JIS
    DISPLAY CHARACTER SET SHIFT_JIS
    AUTHORIZATION RDB$DBHANDLE
    CHARACTER LENGTH CHARACTERS
.
.
.
  INSERT INTO SHIFTJIS_TABLE ( SHIFTJIS_COL1)
   VALUES
        (:LAST_NAME);
.
.
.
$ SQLMOD/LIST=A.LIS A.SQLMOD
.
.
.
        ( SHIFTJIS_COL1)
          1
%SQL−F−INCCSASS, (1) Incompatible character set assignment between
SHIFTJIS_COL1 and :LAST_NAME

$ type A.LIS

.
.
.
Command Line Summary:

    SJIS_MOD2_M.SQLMOD /LIST
```

```
        /G_FLOAT
        /WARN=(WARNING, DEPRECATED)
        /NOFLAG_NONSTANDARD
        /CONSTRAINT_DEFAULT=DEFERRED
        /NOCONNECT
        /INIT_HANDLES
        /NORESTRICT_INVOKER
        /CHECK_RW
        /ANSI_VIEWS
        /ANSI_DATE
        /ANSI_QUOTING
        /ANSI_PARAMETERS
        /QUERY_ESTIMATES
        Default Character Set: DEC_MCS
        National Character Set: SHIFT_JIS
        Identifiers Character Set: SHIFT_JIS
        Literals Character Set: SHIFT_JIS
        Character Length in Characters
.
.
.
```

Note that the Default Character Set as shown in the listing file has not been set correctly.

A workaround for this problem is to use NAMES ARE in the module header to set the desired character set prior to setting the Default Character Set.

```
.
.
.
DECLARE MODULE
    DIALECT SQL92
    NAMES ARE SHIFT_JIS
    DEFAULT CHARACTER SET SHIFT_JIS
    NATIONAL CHARACTER SET SHIFT_JIS
    IDENTIFIER CHARACTER SET SHIFT_JIS
    LITERAL CHARACTER SET SHIFT_JIS
    DISPLAY CHARACTER SET SHIFT_JIS
    AUTHORIZATION RDB$DBHANDLE
    CHARACTER LENGTH CHARACTERS
.
.
.
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.3 Oracle RMU Errors Fixed

4.3.1 RMU Extract Not Formatting View Column Expressions Correctly

Bug 1832240

In prior releases of Oracle Rdb, the RMU Extract command did not correctly format VIEW definitions that contained computed expressions in the SELECT clause, such as that shown below.

```
create view V1 (F3) as
  select sum (F3 /
              case (select cast (F1 as integer) from T1
                    where F2 = 'STR_VALUE')
               when 0 then 1
               when 1 then 10
               when 2 then 100
               when 3 then 1000
               when 4 then 10000
               when 5 then 100000
               else 0
              end)
    from T2;
```

This example was extracted below. Note the incorrect formatting of the expression and the missing separating white space. This made the generated definition illegal.

```
create view "V1"
    (F3) as
    select

     sum((C2.F3 / case (select CAST(C3.F1 AS INTEGER) from T1 C3where (C3.F2 =
    'STR_VALUE')) when 0 then 1 when 1 then 10 when 2 then 100 when 3 then 1000
     when 4 then 10000 when 5 then 100000 else 0end)) from T2 C2;
```

The only workaround for this problem is to manually edit the definition after extracting with RMU Extract or to revert to the original view source.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.3.2 RMU/UNLOAD/AFTER_JOURNAL Fragmented Records Clarification

The RMU /UNLOAD /AFTER_JOURNAL Utility uses additional CPU and memory resources while processing and unloading fragmented records from the after−image journal file. As record fragments are found within a transaction, they are buffered in memory on a "fragment" queue. After all non−fragmented records from the transaction have been output, the fragmented records are reconstructed and output.

Because the fragments are buffered in memory, additional process page file quota may be required when unloading transactions that have a large number of record fragments. Also additional process working set quota may be required in order to limit process page faulting.

4.3.3 RMU/DUMP/BACKUP Did Not Check the VMS BYPASS Privilege

Bug 1966820

The RMU/DUMP/BACKUP command for Oracle Rdb RMU did not check if the user process was granted the VMS BYPASS privilege if the user was not granted the necessary RMU access privileges to the database backup file created by the RMU/BACKUP command. Therefore, the RMU/DUMP/BACKUP command did not execute even though the BYPASS privilege should have allowed the user to execute the command.

The following example shows that even though the BYPASS privilege should have allowed the user to override the lack of RMU privileges to access the backup file, the user was denied access by the RMU/DUMP/BACKUP command.

```
$RMU /DUMP /BACKUP_FILE PERSONNEL
%RMU-I-DMPTXT_163, No dump option selected. Performing read check.
%RMU-F-NOPRIVERR, no privileges for attempted operation
%RMU-F-FTL_DUMP, Fatal error for DUMP operation at 30-AUG-2001 16:42:17.96
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.3.4 RMU/BACKUP Invalid Volume 1 Tape Label When Used With COMPAQ SLS

Bug 1969648

The RMU/BACKUP command for Oracle Rdb RMU, when used with COMPAQ SLS, did not detect the case where SLS did not provide a new VOL1 label to replace the VOL1 label that Rdb RMU/BACKUP was about to write to the first tape volume. RMU/BACKUP therefore wrote an 80 character label buffer to the tape that contained invalid characters. This caused an RMU−F−LABELERR when the tape was restored using RMU/RESTORE.

This problem only happens when RMU/BACKUP is run with COMPAQ SLS and when COMPAQ SLS does not modify the 80 character VOL1 label that RMU/BACKUP writes to the first tape volume.

The following example shows that although the RMU/BACKUP with SLS did not show an error, a VMS DUMP command of the BACKUP tape shows an invalid label on the first backup tape volume. Therefore, RMU/RESTORE returns an RMU−F−LABELERR.

Here is an example of a valid RMU/BACKUP tape label on the first tape volume created after this problem was fixed (note that this is just an example and correct labels may vary).

```
$ DUMP TAPEDEVICE:

Dump of device tapedevice: on 29-AUG-2001 11:44:32.94

Block number 1 (00000001), 80 (0050) bytes

20202020 20202020 20202020 20202020 20202020 20203035 30494241 314C4F56

VOL1ABI050                      000000
```

```
20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020

                           000020

33202020 20202020 20202020 202020203.............. 000040
```

Here is an example of an invalid RMU/BACKUP tape label on the first tape volume that has been created by this problem (note that this is just an example and incorrect labels may vary).

```
$ DUMP TAPEDEVICE:

Dump of device tapedevice: on 29−AUG−2001 ...

Block number 1 (00000001), 80 (0050) bytes

00000000 00001F00 00000000 00183390 FFFFFFFF FFFFFFFE 00000000 000119D8

Ø..............3.............. 000000

00000000 00000000 00000000 007EBFC0 00000000 00183390 00000000 00010DC0

À........3......À¿~.............. 000020

00000000 00000000 00000000 00000D05

................................ 000040
```

Here is an example of the RMU−F−LABELERR returned by RMU/RESTORE.

```
%RMU−F−LABELERR, error in tape label processing on
tapedevice:[000000]SAMPLE_DB.RBF;
−RMU−F−NOTANSI, tape is not valid ANSI format
%RMU−F−FATALERR, fatal error on RESTORE
%RMU−F−FTL_RSTR, Fatal error for RESTORE operation ...
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.3.5 RMU/ANALYZE/CARDINALITY Fails on Databases With Local Temporary Tables

Bug 2019322

RMU/Analyze/Cardinality, when attempting to process LOCAL temporary tables, generated an error and failed to execute.

```
$ RMU /ANALYZE /CARDINALITY SQL$DATABASE
%RDMS−E−BAD_CODE, corruption in the query string
%RMU−F−FATALRDB, Fatal error while accessing Oracle Rdb.
%RMU−F−FTL_ANA, Fatal error for ANALYZE operation at 27−SEP−2001 13:34:25.79
```

RMU has now been corrected to ignore temporary tables as well as views.

The workaround for this problem is to use the RMU/SHOW OPTIMIZER/STATISTIC=CARD command or the RMU/COLLECT OPTIMIZER_STATISTICS command if RMU/ANALYZE/CARDINALITY/UPDATE

was tried.

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.3.6 File Name Not Displayed By RMU /RESTORE for Extend Failure

Bug 1822217

When an RMU /RESTORE operation is unable to extend a storage area, it is possible for the error message displayed to not include the name of the file. This may make it difficult to determine which device has inadequate free space. In the following example note that the name of the file is not displayed.

```
$ RMU /RESTORE ...
%RMU-F-FILACCERR, error extending file
-SYSTEM-W-DEVICEFULL, device full; allocation failure
%RMU-F-FTL_RSTR, Fatal error for RESTORE operation at 17-JUN-2001 03:08:55
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1. RMU /RESTORE now displays the file name, where possible, during a failed file extend operation.

4.3.7 RMU/SHOW STATISTICS Allowed Suspend of Disabled ABS

Previously, the RMU /SHOW STATISTICS Utility allowed the user to suspend AIJ Backup Server (ABS) operations on a node even when the ABS was disabled. This could lead to confusing errors during later manual AIJ backup operations.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. The RMU /SHOW STATISTICS Utility now does not allow the ABS to be suspended when it is not enabled.

4.3.8 RMU/COPY/BLOCKS_PER_PAGE Can Corrupt Copied Database Uniform Areas

Bug 2028181

For the RMU/COPY command, if the "/blocks_per_page" qualifier was not specified for a particular storage area but was for all database storage areas, database corruption of uniform storage areas occurred to the copied database. As documented in the Oracle Rdb RMU Reference Manual for the RMU/COPY command, BLOCKS PER PAGE can only be changed for MIXED storage areas, not UNIFORM storage areas. But when the "/blocks_per_page" qualifier was used for all storage areas, RMU incorrectly bypassed the check for UNIFORM storage areas and attempted to change the BLOCKS PER PAGE setting for UNIFORM as well as MIXED storage areas. This caused the database corruption of the moved copy of the database. Now, the number of BLOCKS PER PAGE will be changed only for MIXED storage areas and a warning message will be output for each UNIFORM storage area that BLOCKS PER PAGE cannot be changed for that area since it is a UNIFORM database storage area.

The following example shows that since /BLOCKS_PER_PAGE=3 was specified for all storage areas in the MF_PERSONNEL database, it caused the database corruption problem for the uniform storage areas in the

copied database.

```
$ RMU /COPY /DIRECTORY=TMPDIR /ROOT=TMPDIR:MFP1 /BLOCKS_PER_PAGE=3 MF_PERSONNEL
%RMU-W-BADPTLARE, invalid larea for uniform data page 5 in storage area 1
%RMU-W-BADPTLAR2,                   SPAM larea_dbid: 16385, page larea_dbid: 1
%RMU-W-BADPTLARE, invalid larea for uniform data page 149 in storage area 1

$ RMU /VERIFY /ALL TMPDIR:MFP1
%RMU-I-BGNROOVER, beginning root verification
%RMU-I-ENDROOVER, completed root verification
%RDB-W-NO_RECORD, access by dbkey failed because dbkey is no longer associated
 with a record
-RDMS-F-NODBK, 61:1179:0 does not point to a data record
%RMU-E-ERRRDBREL, error accessing RDB$RELATIONS relation
```

The following example shows that the problem is now fixed.

```
$ RMU /COPY /DIRECTORY=TMPDIR /ROOT=TMPDIR:MFP1 /BLOCKS_PER_PAGE=3 MF_PERSONNEL
%RMU-W-UNIFORMBLOCKS, BLOCKS PER PAGE cannot be changed for uniform storage
 area RDB$SYSTEM
%RMU-W-UNIFORMBLOCKS, BLOCKS PER PAGE cannot be changed for uniform storage
area MF_PERS_SEGSTR

$ RMU /VERIFY /ALL TMPDIR:MFP1
```

To avoid this problem, specify /BLOCKS_PER_PAGE for each individual storage area in the RMU/COPY command, not as a default for all storage areas.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. A warning message is displayed and the uniform storage area page size is not changed.

4.3.9 DROPped Storage Area and RMU /VERIFY in Cluster

Bug 1421362

Previously, when a database was opened in a cluster environment, it was possible for the RMU /VERIFY command to be unable to open storage area files when storage areas were moved or dropped on another node in the cluster.

For example, consider the following sequence of events on a two node cluster (consisting of NODE1 and NODE2):

```
Node1$: RMU /OPEN MFP

Node2$: RMU /OPEN MFP

Node1$: SQL$ ALTER DATABASE FILENAME MFP DROP STORAGE AREA U1;

Node2$: RMU /VERIFY MFP
    .
    .
    .
%RMU-F-OPNFILERR, error opening file U1.RDA
%RMU-F-FILNOTFND, file not found
%RMU-E-BDAREAOPN, unable to open file U1.RDA  for storage area
%RMU-F-ABORTVER,  fatal error encountered; aborting verification
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1. The RMU /VERIFY Utility now correctly detects storage areas that have been dropped or moved.

4.3.10 RMU /VERIFY Checks All Storage Area Files First

Bug 671681

Previously, the RMU /VERIFY command would abort and return a fatal error to the user when a storage area file was unable to be opened (for example, when the storage area file had been deleted). However, other storage areas were not checked, leading to the possibility that not all problems with missing storage area files were reported.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. The RMU /VERIFY Utility now checks all storage area files and reports problems while opening the files before returning a fatal error. This makes it much easier to know what files must be restored with the RMU /RESTORE command.

4.3.11 RMU/SHOW STATISTICS Multi−Page Report File

Previously, the RMU /SHOW STATISTICS Utility only displayed the first page ("Page: 1 of 1") of multi−page displays. This made it difficult, at times, to find specific information.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. The RMU /SHOW STATISTICS Utility now writes all pages of multi−page displays to the report file. Note that for some screens (storage area information, row cache information, and so on), there can be a significant amount of data written and this can result in a dramatic increase in the size of the report file.

4.3.12 Area Locks Demoted Statistic Not Always Correctly Incremented

Previously, the "locks demoted" statistic for "area" locks was not always correctly incremented. This could occur, for example, when a read−only transaction was started when the previous transaction was a read−write transaction. The "locks promoted" statistic could have been incorrectly incremented in this case. This, in turn, lead to potentially confusing results when comparing the "locks promoted" rate with the "locks demoted" rate for "area" locks in the "RMU/SHOW STATISTICS" facility.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. The correct statistic is now incremented when an "area" lock is demoted from one lock mode to a lower mode.

4.3.13 RMU /BACKUP /ONLINE /NOQUIET_POINT Fails

Oracle Rdb Release 7.1.0 introduced a potential regression where the RMU /BACKUP /ONLINE /NOQUIET_POINT command may fail with an incorrect error message indicating that it is unable to write to the root file. This is an example of the incorrect error from the RMU /BACKUP command:

```
$ RMU /BACKUP /ONLINE /NOQUIET_POINT MFP NLA0:MFP
%RMU−F−FILACCERR, error writing file DUA0:[DB]MFP.RDB;1
%RMU−F−FTL_BCK, Fatal error for BACKUP operation ...
```

This problem has been corrected in Oracle Rdb Release 7.1.0.1.

4.4 LogMiner Errors Fixed

4.4.1 LogMiner Compresses Pre−Delete Record Content

Previously, when the Oracle Rdb LogMiner(TM) feature was enabled, the pre−delete record contents were not compressed prior to being journaled. Because of this, it was possible for AIJ files to grow excessively if many large records were being deleted.

This problem has been corrected in Oracle Rdb Release 7.1.0.1. When the Oracle Rdb LogMiner feature is enabled, pre−delete record contents are now correctly compressed. Because of the difference in pre−delete record contents in an AIJ file, it is important that AIJ files created with prior versions of Oracle Rdb be processed with the matching version of the Oracle Rdb LogMiner (RMU /UNLOAD /AFTER_JOURNAL command).

When using the Oracle Rdb LogMiner feature, existing AIJ files should be backed up and processed prior to upgrading to this release of Oracle Rdb.

Failure to use the correct version of the Oracle Rdb LogMiner to process an AIJ file typically results in RMU−W−RECVERDIF warnings when pre−delete record contents are being processed.

LogMiner AIJ files not compatible

When the Oracle Rdb LogMiner(TM) feature is being used, AIJ files from this version of Oracle Rdb are not compatible with the Oracle Rdb LogMiner feature from prior versions of Oracle Rdb. Only the Oracle Rdb LogMiner feature is affected; AIJ recovery is not affected. If the Oracle Rdb LogMiner feature is not enabled for a database, there is no difference in the format or content of an AIJ file.

4.5 Optimizer Problems Fixed in Oracle Rdb Release 7.1.0.

The following Optimizer Bugs were fixed in Oracle Rdb Release 7.1.0 but the release notes were inadvertently left out.

4.5.1 Query Having OR Compound Predicates With Subquery Returns Wrong Results

Bug 1527102

The following query contains the OR of three predicates: one of which is based on the results of a subquery; one of which is a filter predicate of the form column = literal; and one of which is a constant of the form literal = literal. The query should return 1 row.

```
set flags 'strategy,detail';
select t1.hmcnr from t1 t1
    where t1.ean='5410103914978' and
    (t1.shop_class = (select sho.shop_class from r_shop sho
                      where sho.shop='460')
     or t1.shop_class='A'
     or 'XXX'='460');
Tables:
  0 = t1
  1 = R_SHOP
Cross block of 2 entries
  Cross block entry 1
    Aggregate: (VIA)
    Conjunct: 1.SHOP = '460'
    Conjunct: 'XXX' = '460'
    Get     Retrieval sequentially of relation 1:R_SHOP
  Cross block entry 2
    Conjunct: (0.ean = '5410103914978') AND ((0.shop_class = {subselect}) OR
              (0.shop_class = 'A') OR ('XXX' = '460'))
    Get     Retrieval sequentially of relation 0:t1
HMCNR
 45281
 45134
2 rows selected
```

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

1. A filter predicate is ANDed to an OR compound predicate
2. The OR compound predicate contains a subquery predicate, a couple of filter predicates and a constant predicate

As a workaround, the query works if the constant predicate is removed.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.2 Query Using OR/AND Predicates With EXISTS Clause Returns Wrong Results

Bug 1569972

The following query using AND/OR predicates with an EXISTS clause should return 1 row:

```
set flags 'strategy,detail';

select t1.c1 from t1 t1, t2 t2 where
((t2.c4 = 1 and
  t2.c5 = 5 and
  not exists (select * from t2 t2a                                  <----
              where t2a.c4 = 4 and t2a.c5  = 5)) or                 <----
 (t2.c4  = 4 and t2.c5 = 5))                                        <----
 and t1.c1 = t2.c6                                                  <----
;
Tables:
  0 = T1
  1 = T2
  2 = T2
Cross block of 3 entries
  Cross block entry 1
    Conjunct: {subselect} = 0
      Aggregate-F1: (COUNT-ANY)     Index only retrieval of relation 2:T2
      Index name  T2_H [2:2]
        Key: (2.C4 = 4) AND (2.C5 = 5)
  Cross block entry 2
    Conjunct: (1.C4 = 1) OR (1.C4 = 4)
    Conjunct: 1.C5 = 5
    Conjunct: {subselect} = 0
    Get     Retrieval by index of relation 1:T2
      Index name  T2_H [(2:2)2] Bool
        Key: ((1.C4 = 1) AND (1.C5 = 5)) OR ((1.C4 = 4) AND (1.C5 = 5))
        Bool: 1.C5 = 5
  Cross block entry 3
    Index only retrieval of relation 0:T1
      Index name  T1_H [1:1]
        Key: 0.C1 = 1.C6
0 rows selected
```

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

1. OR parent predicate with AND predicates on each branch
2. One of the OR branches also includes a subquery, such as NOT EXISTS
3. A second AND predicate is appended after the OR parent predicate

As a workaround, the problem can be corrected if you move the second AND predicate to the front of the OR parent predicate, as follows:

```
set flags 'strategy,detail';

select t1.c1 from t1 t1, t2 t2 where
t1.c1 = t2.c6 and                                                   <----
((t2.c4 = 1 and
  t2.c5 = 5 and
```

```
  not exists (select * from t2 t2a                                  <----
              where t2a.c4 = 4 and t2a.c5  = 5)) or                 <----
 (t2.c4  = 4 and t2.c5 = 5))                                        <----
;
```

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.3 Query Using German Collating Sequence Returns Wrong Results

Bug 1530947

The following query, in a database where the German Collating Sequence is used by default, returns wrong results (should return some rows):

```
SELECT  p.datum, p.produkt, p.abtlg, p.stelle
 FROM v_team_datum p,
      produkte g
 where
       p.abtlg=g.abtlg  ;
Conjunct
Match
  Outer loop
    Sort    Conjunct        Aggregate       Sort    Conjunct
    Leaf#01 BgrOnly PROD_DATEN Card=24063
      BgrNdx1 IDX_PROD_DATEN_SORT [1:1] Fan=8
  Inner loop      (zig-zag)
    Conjunct        Get     Retrieval by index of relation PRODUKTE
      Index name  IDX_PRODUKTE_SORT [0:0]
0 rows selected
```

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

1. The query is a simple join between a view and one table, with the join predicate of CHAR data type
2. The optimizer uses a match strategy to join them, where a comparison of the join keys requires the process of encoding the CHAR data type into the German collating sequence

As a workaround, the query works if a view with the same attributes as the table is used instead of the table itself, as in the following example:

```
SELECT  p.datum, p.produkt, p.abtlg, p.stelle
 FROM v_team_datum p,
      view_produkte g
 where
       p.abtlg=g.abtlg  ;
Cross block of 2 entries
  Cross block entry 1
    Conjunct        Aggregate       Sort    Conjunct
    Leaf#01 BgrOnly PROD_DATEN Card=24063
      BgrNdx1 IDX_PROD_DATEN_SORT [1:1] Fan=8
  Cross block entry 2
    Leaf#02 FFirst PRODUKTE Card=25
      BgrNdx1 IDX_PRODUKTE_SORT [3:3] Fan=6
```

The query works because the optimizer applies a cross strategy instead of a match strategy.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.4 Left Outer Join Query Returns Wrong Results When ON Clause Evaluates to False

Bug 1581632

The following left outer join query returns wrong results when the join conditions in the ON clause evaluate to false for all rows:

```
set flags 'strategy,detail';
select tt.employee_id, tt.last_name, jh.job_code
from
    (select e.employee_id, e.last_name
     from degrees d, employees e where
        e.employee_id = '00354'
        and d.employee_id = e.employee_id) as tt
    left outer join
    job_history jh
      on tt.last_name = '?' and                          <----
         jh.job_code = tt.employee_id;                   <----
Tables:
  0 = DEGREES
  1 = EMPLOYEES
  2 = JOB_HISTORY
Cross block of 2 entries        (Left Outer Join)
  Cross block entry 1
    Conjunct: "tt.last_name" = '?'
    Merge of 1 entries
      Merge block entry 1
      Cross block of 2 entries
        Cross block entry 1
          Get     Retrieval by index of relation 1:EMPLOYEES
            Index name  EMPLOYEES_HASH [1:1]       Direct lookup
              Key: 1.EMPLOYEE_ID = '00354'
        Cross block entry 2
          Index only retrieval of relation 0:DEGREES
            Index name  DEG_EMP_ID [1:1]
              Key: 0.EMPLOYEE_ID = 1.EMPLOYEE_ID
  Cross block entry 2
    Conjunct: ("tt.last_name" = '?') AND
              (2.JOB_CODE = tt.employee_id)
    Get     Retrieval by index of relation 2:JOB_HISTORY
      Index name  JH_EMPLOYEE_ID [0:0]
0 rows selected
```

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

1. Left outer join query on a subquery and job_history of mf_personnel database
2. ON clause containing two or more predicates, and the ON clause evaluates to false for all rows, for example:

   ```
   "last_name" = '?' and jh.job_code = tt.employee_id
   ```

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.5 Query With Two IN Clauses on Two Subqueries Returns Wrong Results

Bug 1585429

The following query with two IN clauses on two subqueries with different match keys, returns a count of 0 when it should return a non−0 count:

```
SELECT count(*) FROM  t1
WHERE
  subclass_id IN (SELECT  DISTINCT subclass_id
                       FROM  t2
                       WHERE  class_id = 'CAJ_C01#')
  AND
  recipe_id   IN (SELECT  recipe_id
                       FROM  t3
                       WHERE  eqp_id = 'CAR−02C'
                      )
;
Aggregate       Conjunct
Match
  Outer loop
    Conjunct
    Match
      Outer loop
        Get     Retrieval by index of relation t1
          Index name  t1_ndx [0:0]
      Inner loop      (zig−zag)
        Aggregate−F1    Conjunct
        Index only retrieval of relation t2
          Index name  t2_ndx [0:0]
  Inner loop      (zig−zag)
    Aggregate−F1    Conjunct        Get
    Retrieval by index of relation t3
      Index name  t3_ndx [1:1]

          0
1 row selected
```

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

1. Two different IN clauses on two subqueries, with different match keys
2. The query applies a match strategy where the outer leg uses the match key (subclass_id) of another match stream that is different from the other key (recipe_id) of the inner leg without sorting the results of the outer leg using the match key (subclass_id).

Oracle Rdb7 Release 7.0.5 applies a sort node on the outer leg and thus returns the correct results.

As a workaround, use a query outline to change the strategy to cross from match.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.6 Query Having Same SUBSTRINGs Within CASE Expression Returns Wrong Results

Bugs 1489972, 1485656, 975091

The following queries, containing the same SUBSTRING expressions within a CASE expression, return wrong results.

The following example shows two simple queries (from Bug 1485656 and Bug 975091) having the same subexpression (SUBSTRING) appearing more than once within the CASE expression. The query in the case of Bug 1489972 is more complicated and thus omitted here. It contains unions of several subselect queries with nested views and SUBSTRING/CASE expressions.

```
! Bug 1485656
!  should return the value 1 for the content of y
! ~Xt: Content of y = 1
!
set FLAGS 'TRACE'
declare :x char(2);
declare :y char(1);
begin
set :x='21';
set :y= case
          when  ((substring(:x from 1 for 1)='1') and
                 (substring(:x from 2 for 1)='1') )
                then 'O'
          else
                (substring(:x from 2 for 1))
        end;
trace 'Content of y = ', :y ;
end;
The output is:
~Xt: Content of y =

! Bug 975091
!  should return the value of 295 for the column RESP
!
create table t1 (c1 char(12));
insert into t1 value ( '29500000199');

select substring( c1 from 1 for 3) ress,
       case
          when 'a' = 'c' and (substring(c1 from 1 for 3)) = '295'
            then 'a'
          when 'c' =  'c'
            then (substring(c1 from 1 for 3))
          else  ' '
       end resp
  from t1;
RESS   RESP
295
1 row selected
```

The key parts of these queries which contributed to the situation leading to the errors are these:

1. CASE expression contains several similar expressions

2. The expression in the WHEN clause is shared in the same clause of another WHEN clause (in the case of Bug 975091)
3. The expression in the WHEN clause is shared in another part of the CASE statement, such as an ELSE clause (in the case of Bug 1485656)

In the case of Bug 1485656, a workaround is to use an IF instead of a CASE statement to get the correct results:

```
set FLAGS 'TRACE'
declare :x char(2);
declare :y char(1);
begin
set :x='21';
        if  ((substring(:x from 1 for 1)='1') and
             (substring(:x from 2 for 1)='1') )
            then
                set :y='O';
        else
                set :y=(substring(:x from 2 for 1));
        end if;
trace 'Content of y:',:y;
end;
```

Another workaround is to use temporary variables for the substrings.

In the case of Bug 975091, the workaround is to swap the WHEN clauses, as in the following example:

```
select substring( c1 from 1 for 3) ress,
       case
          when 'c' =  'c'
            then (substring(c1 from 1 for 3))                      ! <= 1st
          when 'a' = 'c' and (substring(c1 from 1 for 3)) = '295' ! <= 2nd
            then 'a'
          else  ' '
       end resp
  from t1;
```

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.7 Aggregate Query With Nested MIN Function Returns Wrong Results

Bug 1408892

The following query should return the value of ADMN for min(d1.department_code):

```
create index dept_managerid_code_ndx on departments
    (manager_id,department_code);

select min(d1.department_code),
    min((select min (d2.department_code)
        from departments d2
        where d1.manager_id = d2.manager_id AND
              d2.budget_actual > 0))
    from departments d1;
```

```
                NULL          NULL
1 row selected
```

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

1. The subselect query has "where" predicates which cause the query to return 0 rows, e.g. "d2.budget_actual > 0"
2. The subselect query contains an aggregate function, e.g. MIN
3. The subselect query is wrapped inside another aggregate function, e.g. MIN

As a workaround to this problem, the query works if the MIN function is removed from the column 'd2.department_code' in the inner subselect, as seen in the following example.

```
select min(d1.department_code),
    min((select d2.department_code
        from departments d2
        where d1.manager_id = d2.manager_id AND
              d2.budget_actual > 0))
    from departments d1;
```

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.8 Query with UNION Subselect Returns Wrong Results

Bug 1656974

The following query with UNION subselect should return 0 rows.

```
set flags 'strategy,detail';
select ps.id, ps.kbn, ps.ymd
 from (select ps1.id,
              ps1.kbn,
              '99999999'                    ! <== this causes the problem
         from  ps ps1, pm pm
         where pm.id = ps1.id
       union all
       select ps2.id,
              ps2.kbn,
              ps2.end_ymd
         from  ps ps2, pm pm
         where pm.id = ps2.id)

      as ps (id, kbn, ymd)
  where ps.id = '021023307' and
        ps.ymd > '12345678' and
        ps.kbn in ('1','2') ;
Tables:
  0 = PS
  1 = PM
  2 = PS
  3 = PM
Merge of 1 entries
  Merge block entry 1
  Merge of 2 entries
    Merge block entry 1
    Conjunct: 1.id = 0.ID
    Match
```

```
      Outer loop      (zig-zag)
        Conjunct: 0.ID = '021023307'
        Conjunct: '99999999' > '12345678'
        Get     Retrieval by index of relation 0:PS
          Index name  IDX_PS_2 [1:1] Bool
            Key: <mapped field> = '021023307'
            Bool: '99999999' > '12345678'
      Inner loop      (zig-zag)
        Index only retrieval of relation 1:PM
          Index name  IDX_PM_0 [0:0]
    Merge block entry 2
    Conjunct: 3.id = 2.ID
    Match
      Outer loop      (zig-zag)
        Conjunct: (2.ID = '021023307') AND (2.end_ymd > '12345678')
                    AND ((2.kbn = '1') OR (2.kbn = '2'))
        Get     Retrieval by index of relation 2:PS
          Index name  IDX_PS_2 [2:1]
            Key: (<mapped field> = '021023307') AND (<mapped field> > '12345678'
                 )
      Inner loop      (zig-zag)
        Index only retrieval of relation 3:PM
          Index name  IDX_PM_0 [0:0]
 ID          KBN   YMD
 021023307   0     99999999
1 row selected
```

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

1. The query contains a subselect of a UNION, where one of the columns is a literal, e.g. '99999999'.
2. The where clause contains an equality predicate, a GTR predicate, and an IN clause.

As a workaround, the query works if the IN clause is moved before the GTR predicate, as in the following example.

```
set flags 'strategy,detail';
! The following query should return 0 rows
!
select ps.ID, ps.kbn, ps.ymd
 from (select ps1.ID,
              ps1.kbn,
              '99999999'
         from  ps ps1, pm pm
         where pm.id = ps1.ID
       union all
       select ps2.id,
              ps2.kbn,
              ps2.end_ymd
         from  ps ps2, pm pm
         where pm.id = ps2.id)

      as ps (id, kbn, ymd)
  where ps.id = '021023307' and
        ps.kbn in ('1','2') and         <=== moved
        ps.ymd > '12345678' ;
```

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.9 Query with CONCATENATE in BETWEEN Clause Returns Wrong Results

Bug 1663038

The following query uses the CONCATENATE function in the BETWEEN clause. It should return 3 rows, but it returns only 1 row.

```
SQL> sh tab ORDER;
Information for table ORDER

 Columns for table ORDER:
 Column Name                     Data Type        Domain
 -----------                     ---------        ------
 ORDER_NO                        CHAR(4)
  Not Null constraint ORDER_NO_NOT_NULL
 SHIP_DATE              CHAR(8)
  Not Null constraint ORDER_NOT_NULL
 SHIP_STAT              CHAR(1)
  Not Null constraint ORDER_NOT_NULL
...etc...

Table constraints for ORDER:
ORDER_NOT_NULL
 Not Null constraint
 Column constraint for ORDER.SHIP_DATE
 Evaluated on COMMIT
 Source:
        ORDER.SHIP_DATE NOT null
...etc...

SQL> sel order_no from customer;
 ORDER_NO
 1ED0
 1j80
 1a78
3 rows selected
SQL> sel order_no,ship_date,ship_stat from order;
 ORDER_NO   SHIP_DATE   SHIP_STAT
 1ED0       20010301    b
 1a78       20010228    a
 1j80       20010301    a
3 rows selected

set flags 'strategy,detail';
set flags 'max_stab';
select  a.order_no, a.ship_date, a.ship_stat
from    ORDER a, CUSTOMER b
where   a.order_no = b.order_no and
        ((a.SHIP_DATE || a.SHIP_STAT)
                BETWEEN '20010228a' '20010301d') ;
Tables:
  0 = ORDER
  1 = CUSTOMER
Cross block of 2 entries
  Cross block entry 1
    Conjunct:
```

```
    (0.SHIP_DATE > SUBSTRING ('20010228a' FROM 0 FOR 8)) OR
     ((0.SHIP_DATE = SUBSTRING ('20010228a' FROM 0 FOR 8)) AND
      (0.SHIP_STAT >= SUBSTRING ('20010228a' FROM 8)))
    Conjunct:
    ((0.SHIP_DATE < SUBSTRING ('20010301d' FROM 0 FOR 8)) AND
     NOT MISSING (0.SHIP_STAT)) OR
     ((0.SHIP_DATE = SUBSTRING ('20010301d' FROM 0 FOR 8)) AND
      (0.SHIP_STAT <= SUBSTRING ('20010301d' FROM 8)))
    Get     Retrieval by index of relation 0:ORDER
      Index name  ORDER_UM01 [0:0]
  Cross block entry 2
    Index only retrieval of relation 1:CUSTOMER
      Index name  CUSTOMER_UM01 [1:1]         Direct lookup
        Key: 0.ORDER_NO = 1.ORDER_NO
 A.ORDER_NO   A.SHIP_DATE   A.SHIP_STAT
 1a78         20010228              a
1 row selected
```

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

1. The table columns contain NOT NULL constraints.
2. The query contains a BETWEEN clause with CONCATENATE function on two columns.

As a workaround, the query works if the column constraint ORDER_NOT_NULL is removed from the columns of table ORDER.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.10 ORDER BY Query With GROUP BY on Two Joined Derived Tables Returns Wrong Results

Bug 1694233

The following query with GROUP BY and ORDER BY clauses on two joined derived tables returns the results in the wrong order.

```
set flags 'strategy,detail';

select
        cast (a.name as char(5)) as name,
        a.datum
  from (select name, datum,
               cast (count (*) as integer) as count_a
          from a
          group by name, datum) a
       join
       (select name, datum,
               cast (count (*) as integer) as count_b
          from b
          group by name, datum) b
       on     a.name = b.name
          and a.datum = b.datum
   group by a.name, b.name, a.datum, b.datum, count_a
   order by name desc, a.datum asc
  ;
Tables:
```

```
  0 = A
  1 = B
Reduce: 0.NAME, 0.DATUM, 1.NAME, 1.DATUM, CAST (<mapped field> AS INT)
Sort: 0.NAME(a), 0.DATUM(a), 1.NAME(a), 1.DATUM(a), CAST (<mapped field> AS INT)
      (a)
Cross block of 2 entries
  Cross block entry 1
    Merge of 1 entries
      Merge block entry 1
      Aggregate: COUNT (*)
      Sort: 0.NAME(a), 0.DATUM(a)
      Get Retrieval sequentially of relation 0:A
  Cross block entry 2
    Merge of 1 entries
      Merge block entry 1
      Aggregate: COUNT (*)
      Sort: 1.NAME(a), 1.DATUM(a)
      Conjunct: (0.NAME = 1.NAME) AND (0.DATUM = 1.DATUM)
      Get     Retrieval sequentially of relation 1:B
 A.NAME   A.DATUM
 AAAA      1-JAN-2000 00:00:00.00   <=== BBBB should be followed by AAAA
 BBBB      1-JAN-2000 00:00:00.00
2 rows selected
```

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

1. The main query contains a GROUP BY clause on the columns of the two joined derived tables with GROUP BY.
2. One of the columns from the derived tables is cast as the same data type.
3. The ORDER BY clause references the cast column but using descending order.

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.11 Left Outer Join Query With CONCATENATE Returns Wrong Results

Bug 1680135

The following left OJ query with CONCATENATE should return 1 row but instead returns 0 rows.

```
set flags 'strategy,detail';
SELECT ttt.entity_id,
       ttt.cpty_id,
       ttt.trade_count
FROM (SELECT tt.entity_id,
             tt.cpty_id,
             SUM (tt.trade_count) as trade_count
      FROM (SELECT df.entity_id,
                   df.cpty_id,
                   case
                     when df.deal_status = 'X' then 1 else 0
                   end as trade_count
            from deal_folder df) as tt
      GROUP BY tt.entity_id, tt.cpty_id) as ttt
```

```
    LEFT OUTER JOIN
     contact c ON (c.cpty_id = ttt.cpty_id)
 WHERE
    ttt.trade_count <> 0
    and ttt.entity_id || ttt.cpty_id  > ''  ! <== this is causing problem
    ;
Tables:
  0 = DEAL_FOLDER
  1 = CONTACT
Conjunct: (<mapped field> <> 0) AND ((0.ENTITY_ID || 0.CPTY_ID) > '') <=(1)
Cross block of 2 entries        (Left Outer Join)
  Cross block entry 1
    Conjunct: <mapped field> <> 0
    Merge of 1 entries
      Merge block entry 1
      Aggregate: SUM (CASE (WHEN (0.DEAL_STATUS = 'X') THEN 1
                  ELSE 0))
      Sort: 0.ENTITY_ID(a), 0.CPTY_ID(a)
      Merge of 1 entries
        Merge block entry 1
        Conjunct: (0.ENTITY_ID || 0.CPTY_ID) > ''
        Index only retrieval of relation 0:DEAL_FOLDER
          Index name  DEAL_FOLDER_MONITOR_IDX [0:0]
  Cross block entry 2
    Conjunct: (<mapped field> <> 0) AND ((0.ENTITY_ID || 0.CPTY_ID) > '') <=(2)
    Conjunct: 1.CPTY_ID = 0.CPTY_ID
    Index only retrieval of relation 1:CONTACT
      Index name  CONTACT_IDX [0:0]
0 rows selected
```

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

1. The main query is a left outer join between a derived table and a table.
2. The derived table contains a GROUP BY clause on the columns of another derived table with an aggregate function SUM as the output column.
3. The main query has a WHERE predicate containing the CONCATENATE function on two or more columns of the derived table.
4. The main query has another WHERE predicate which references the output column of the aggregate function from the derived table.

As a workaround, the query works if the table 1:CONTACT has some rows or the following CONCATENATE function is replaced by the following predicates:

```
    ttt.entity_id || ttt.cpty_id  > ''

    is replaced by

    ttt.entity_id > '' AND ttt.cpty_id > ''
```

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.12 Query With UNION in German Collating Sequence Returns Wrong Results

Bug 1684612

The following query with a UNION clause, in a database where the German Collating Sequence is used by default, returns wrong results (it should return some rows).

```
select d.datum, d.id, d.team
from teamer d,
      (select s.datum,s.id, s.team
        from team_datum s
        union all
       select datum, id, team
        from team_datum
          ) as s
where
    d.datum=s.datum
      ;
Tables:
  0 = teamer
  1 = team_datum
  2 = team_datum
Conjunct: 0.datum = <mapped field>
Match
  Outer loop
    Sort: <mapped field>(a)
    Merge of 1 entries
      Merge block entry 1
      Merge of 2 entries
        Merge block entry 1
        Get Retrieval sequentially of relation 1:team_datum
        Merge block entry 2
        Get Retrieval sequentially of relation 2:team_datum
  Inner loop
    Temporary relation
    Sort: <mapped field>(a)
    Get Retrieval sequentially of relation 0:teamer
0 rows selected
```

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

1. The query is a simple join between a table and a derived table of subselects unioned together.
2. The join predicate uses CHAR data type.
3. The Optimizer uses a match strategy to join them, where a comparison of the join keys requires the process of encoding the CHAR data type into German collating sequence.

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.13 Query With OR Predicate on Aggregate Column Returns Wrong Results

Bugs 1708342 and 1721323

Query #1:

The following query with an OR predicate should return 1 row with T1.STATUS = 3 but returns an extra row with T1.STATUS = 5. This row does not satisfy the condition in the predicate "x.summe is null".

```
set flags 'max_stability';
set flags 'strategy,detail';
select
    t1.id,
    t1.status,
    t1.anzahl_stuecke,
    x.summe
  from table1 t1,
    (select sum(anzahl_stuecke) as summe
     from table2 t2
     where t1.id = t2.id ) x
  where
     t1.status = 3
     OR
     (t1.status = 5 and  x.summe is null) ;
Tables:
  0 = TABLE1
  1 = TABLE2
Cross block of 2 entries
  Cross block entry 1
    Conjunct: (0.STATUS = 3) OR (0.STATUS = 5)
    Get     Retrieval by index of relation 0:TABLE1
      Index name  XPKTABLE1 [0:0]
  Cross block entry 2
    Merge of 1 entries
      Merge block entry 1
      Aggregate: SUM (1.ANZAHL_STUECKE)
      Get     Retrieval by index of relation 1:TABLE2
        Index name  XPKTABLE2 [1:1]
          Keys: 0.ID = 1.ID
       T1.ID     T1.STATUS   T1.ANZAHL_STUECKE                X.SUMME
           1             3                  10                   NULL
           2             5                  10                     10
2 rows selected
```

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

1. The main query joins a table and a derived table with a column of an aggregate function (e.g. SUM).
2. The WHERE clause contains an OR predicate, where one of the branches references the aggregated column.

As a workaround, the query works if the branches of the OR predicates are swapped, as in the following example.

```
select
    t1.id,
    t1.status,
    t1.anzahl_stuecke,
    x.summe
  from table1 t1,
    (select sum(anzahl_stuecke) as summe
     from table2 t2
     where t1.id = t2.id ) x
  where
     (t1.status = 5 and  x.summe is null)
     OR
     t1.status = 3 ;
Tables:
```

```
  0 = TABLE1
  1 = TABLE2
Cross block of 2 entries
  Cross block entry 1
    Get     Retrieval by index of relation 0:TABLE1
      Index name  XPKTABLE1 [0:0]
  Cross block entry 2
    Conjunct: ((0.STATUS = 5) AND MISSING (var) OR (0.STATUS = 3)
    Merge of 1 entries
      Merge block entry 1
      Aggregate: SUM (1.ANZAHL_STUECKE)
      Get     Retrieval by index of relation 1:TABLE2
        Index name  XPKTABLE2 [1:1]
          Keys: 0.ID = 1.ID
       T1.ID     T1.STATUS   T1.ANZAHL_STUECKE                X.SUMME
           1             3                  10                   NULL
1 row selected
```

Query #2:

The following query with an OR predicate should return 0 rows.

```
set flags 'max_stability';
set flags 'strategy,detail';
select
    t1.id,
    t1.status,
    t1.anzahl_stuecke,
    x.summe
  from table1 t1,
    (select
        sum(anzahl_stuecke) as summe,
        'hello' as Artikel
     from table2 t2
     where t1.id = t2.id ) x
  where
    t1.id <> 5 and
     x.Artikel = 'hello should not be found' and
    ((t1.status =3) or
      (t1.status = 5 and (x.summe is NULL))
    );
Tables:
  0 = TABLE1
  1 = TABLE2
Cross block of 2 entries
  Cross block entry 1
    Get     Retrieval by index of relation 0:TABLE1
      Index name  XPKTABLE1 [0:0]
        Bool: 0.ID <> 5
  Cross block entry 2
    Conjunct: (0.STATUS = 3) OR ((0.STATUS = 5) AND MISSING (var)
    Merge of 1 entries
      Merge block entry 1
      Aggregate: SUM (1.ANZAHL_STUECKE)
      Get     Retrieval by index of relation 1:TABLE2
        Index name  XPKTABLE2 [1:1]
          Keys: 0.ID = 1.ID
          Bool: (1.ID <> 5) AND ('hello' = 'hello should not be found')
       T1.ID     T1.STATUS   T1.ANZAHL_STUECKE                X.SUMME
           1             3                  10                   NULL
```

```
           2             5                  10                   NULL
2 rows selected
```

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

1. The main query joins a table and a derived table with the column of an aggregate function (e.g. SUM) and a column of a constant string.
2. The WHERE clause contains an OR predicate, where one of the branches references the aggregate column.
3. The WHERE clause contains additional AND predicates where one of them references the column of a constant string.

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.14 Query With Equality Predicate Included in IN Clause Returns Wrong Results

Bug 1727181

The following query with an equality predicate included in the IN clause should find the row.

```
set flags 'strategy,detail';
sel employee_id
  from employees e, departments d
  where
    e.employee_id = d.manager_id and
    d.department_code in ('ADMN', 'ENG', 'MKTG') and
    d.department_code = 'ENG'
    ;
Tables:
  0 = EMPLOYEES
  1 = DEPARTMENTS
Cross block of 2 entries
  Cross block entry 1
    Conjunct: (1.DEPARTMENT_CODE = 'ADMN') OR (1.DEPARTMENT_CODE = 'MKTG')
    Conjunct: 1.DEPARTMENT_CODE = 'ENG'
    Index only retrieval of relation 1:DEPARTMENTS
      Index name  DEPT_DEPTCODE_MGRID [1:1]
        Keys: 1.DEPARTMENT_CODE = 'ENG'
  Cross block entry 2
    Index only retrieval of relation 0:EMPLOYEES
      Index name  EMP_EMPID_STATUS_CODE [1:1]
        Keys: 0.EMPLOYEE_ID = 1.MANAGER_ID
0 rows selected
```

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

1. The query joins two tables using a join predicate.
2. The query has an equality predicate which is also included in the IN clause.

As a workaround, the query works if the equality predicate is moved to the front of the IN clause, as in the following example.

```
set flags 'strategy,detail';
sel employee_id
  from employees e, departments d
  where
    e.employee_id = d.manager_id and
    d.department_code = 'ENG' and               <== move to front
    d.department_code in ('ADMN', 'ENG', 'MKTG')
    ;
Tables:
  0 = EMPLOYEES
  1 = DEPARTMENTS
Cross block of 2 entries
  Cross block entry 1
    Conjunct: 1.DEPARTMENT_CODE = 'ENG'
    Conjunct: (1.DEPARTMENT_CODE = 'ADMN') OR (1.DEPARTMENT_CODE = 'ENG') OR (
              1.DEPARTMENT_CODE = 'MKTG')
    Index only retrieval of relation 1:DEPARTMENTS
      Index name  DEPT_DEPTCODE_MGRID [1:1]
        Keys: 1.DEPARTMENT_CODE = 'ENG'
  Cross block entry 2
    Conjunct: 1.DEPARTMENT_CODE = 'ENG'
    Index only retrieval of relation 0:EMPLOYEES
      Index name  EMP_EMPID_STATUS_CODE [1:1]
        Keys: 0.EMPLOYEE_ID = 1.MANAGER_ID
 E.EMPLOYEE_ID
 00471
1 row selected
```

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.15 Match Strategy on Columns of Different Size, Using Collating Sequence, Returns Wrong Results

Bug 1684643

The following query using match strategy on columns of different size, using German collating sequence, should find the row.

```
select d.datum, d.abtlg, d.team, d.art
from teamergebnis_kumul d,
      (select m.datum,m.abtlg, m.art, m.team
       from std_team_datum m, prod_kumul_datum v
             where m.datum=v.datum and
            m.abtlg=v.abtlg and
            m.team=v.produkt AND
            m.team='11.3512'
        group by m.datum, m.abtlg, m.art, m.team) AS
        s (datum, abtlg, art, team)
where d.datum=s.datum and
      d.abtlg=s.abtlg and
      d.team=s.team and
      d.art=s.art and
      d.abtlg='465' and d.datum='20001031' and
```

```
      d.team='11.3512';
Tables:
  0 = TEAMERGEBNIS_KUMUL
  1 = STD_TEAM_DATUM
  2 = PROD_KUMUL_DATUM
Cross block of 2 entries
  Cross block entry 1
    Conjunct: 0.TEAM = '11.3512'
    Get     Retrieval by index of relation 0:TEAMERGEBNIS_KUMUL
      Index name  IDX_TEAMERGEBNIS_KUMUL_SORT [3:3]
        Keys: (0.TEAM = '11.3512') AND (0.DATUM = '20001031') AND (0.ABTLG =
              '465')
  Cross block entry 2
    Conjunct: 0.ABTLG = 1.ABTLG
    Conjunct: 0.TEAM = 1.TEAM
    Conjunct: 0.ART = 1.ART
    Merge of 1 entries
      Merge block entry 1
      Reduce: 1.TEAM, 1.ABTLG, 1.DATUM, 1.ART
      Sort: 1.TEAM(a), 1.ABTLG(a), 1.DATUM(a), 1.ART(a)
      Conjunct: (1.DATUM = 2.DATUM) AND (1.ABTLG = 2.ABTLG) AND (1.TEAM =
                2.PRODUKT)
      Match
        Outer loop
          Sort: 1.TEAM(a), 1.ABTLG(a), 1.DATUM(a)
          Conjunct: 1.TEAM = '11.3512'
          Get     Retrieval by index of relation 1:STD_TEAM_DATUM
            Index name  IDX_STD_TEAM_DATUM_SORT [2:2]
              Keys: (0.DATUM = 1.DATUM) AND (1.ABTLG = '465')
        Inner loop
          Temporary relation
          Sort: 2.PRODUKT(a), 2.ABTLG(a), 2.DATUM(a)
          Conjunct: 2.PRODUKT = '11.3512'
          Get     Retrieval by index of relation 2:PROD_KUMUL_DATUM
            Index name  IDX_PROD_KUMUL_DATUM_SORT [2:2]
              Keys: (2.DATUM = 0.DATUM) AND (2.ABTLG = '465')
0 rows selected
```

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

1. The main query is a simple join between a table and a derived table of subselect subquery, joining two tables using 3 equality predicates.
2. The join predicate uses columns of CHAR data type but different column size.
3. The optimizer uses a match strategy to join them, where a comparison of the join keys requires the process of encoding the CHAR data type into German collating sequence.

As a workaround, the query works if the match strategy is changed to use cross by using an outline.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.16 Left Outer Join Query With CAST Function on USING Column Bugchecks

Bug 1802653

The following left outer join query with CAST function on USING column bugchecks.

```
select count(*) from
( select p.paketwert from
  ( select
        cast(packet as integer)   ! <=== CAST causing bugcheck
    from
    serien inner join sujet using (sujet)
  ) as p (paketwert)
) as astpreis (paketwert)
left outer join
( select t.paketwert from
  ( select
        packet
    from
    serien inner join sujet using (sujet)
  ) as t (paketwert)
) as opt(paketwert)
USING (paketwert) ;
```

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

1. The main query is a left outer join of 2 nested derived tables.
2. The CAST function is placed on the column of USING clause.

There is no known workaround for this problem.

This problem has been corrected in Oracle Rdb Release 7.1.0.

4.5.17 Query Using Constant Values in OR Predicates Returns Wrong Results

Bug 1769447

The following query using constant values in OR predicates should return 3 rows.

```
set flags 'strategy,detail';

SELECT  col1 FROM
 (SELECT
     t2.col1 as col1,
     t2.col2 as col2,
     t2.col3 as col3
     from table1 t1, table2 t2
     where t1.col1_id = t2.col1_id
     ) as
   vt (col1, col2, col3)
WHERE
        vt.col3 > 0  AND
        vt.col2 >= 0       AND
        ( vt.col1 <= 3 OR 'hostvar' = 'foo' );
Tables:
  0 = TABLE1
  1 = TABLE2
Merge of 1 entries
  Merge block entry 1
  Conjunct: 0.col1_id = 1.col1_id
  Match
```

```
    Outer loop      (zig-zag)
      Index only retrieval of relation 0:TABLE1
        Index name  TABLE1_NDX [0:0]
    Inner loop      (zig-zag)
      Conjunct: (1.col3 > 0) AND (1.col2 >= 0)
      Get     Retrieval by index of relation 1:TABLE2
        Index name  TABLE2_NDX [0:0]
          Bool: <error: common keyonly boolean no predicates>
        COL1
           1
           2
           3
           4
           5
           6
6 rows selected
```

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

1. The query selects from a derived table of a subselect joining 2 tables.
2. The WHERE clause contains 2 AND predicates and 1 OR predicate.
3. The OR predicate contains a branch of constant predicates, such as "1 = 2".

As a workaround, the query works if the constant condition "'hostvar' = 'foo'" is omitted, as in the following example.

```
set flags 'strategy,detail';

SELECT  col1 from
 (SELECT
     t2.col1 as col1,
     t2.col2 as col2,
     t2.col3 as col3
     from table1 t1, table2 t2
     where t1.col1_id = t2.col1_id
     ) as
   vt (col1, col2, col3)
WHERE
        vt.col3 > 0  AND
        vt.col2 >= 0       AND
        ( vt.col1 <= 3
!           OR 'hostvar' = 'foo'              <=== commented out
            );
Tables:
  0 = TABLE1
  1 = TABLE2
Merge of 1 entries
  Merge block entry 1
  Conjunct: 0.col1_id = 1.col1_id
  Match
    Outer loop      (zig-zag)
      Index only retrieval of relation 0:TABLE1
        Index name  TABLE1_NDX [0:0]
    Inner loop      (zig-zag)
      Conjunct: (1.col3 > 0) AND (1.col2 >= 0) AND (1.col1 <= 3)
      Get     Retrieval by index of relation 1:TABLE2
        Index name  TABLE2_NDX [0:0]
          Bool: 1.col1 <= 3
        COL1
```

```
           1
           2
           3
3 rows selected
```

This problem has been corrected in Oracle Rdb Release 7.1.0.

Chapter 5
Enhancements

5.1 Enhancements Provided in Oracle Rdb Release 7.1.0.2

5.1.1 Buffer Objects Enhancements

In Release 6.1, Oracle Rdb introduced minimal support for OpenVMS Fast I/O Buffer Objects and the Fast I/O feature. Prior to Release 7.1, Oracle Rdb users who wanted to utilize Fast I/O could define the following logical name to have the Oracle Rdb I/O data buffers turned into a buffer objects:

```
$ DEFINE RDM$BIND_BUFOBJ_ENABLED 1
```

This logical name is no longer used; new database parameters and logical names have been introduced to offer more control over enabling buffer objects for various Oracle Rdb buffers.

A new command "RMU /SET BUFFER_OBJECT [/LOG] [/ENABLE=...] [/DISABLE=...] database" is available to control, on a database basis, which database objects are to use the OpenVMS Fast I/O and Buffer Objects features. This command accepts the "/ENABLE=(...)" and "/DISABLE=(...)" qualifiers. Specify the keywords PAGE, AIJ, RUJ or ROOT to enable or disable buffer objects. If a keyword is specified in both the "/ENABLE" and "/DISABLE" qualifiers, the "/ENABLE" qualifier overrides the "/DISABLE" qualifier and the buffer object state is enabled for the specified object type.

Table 5−1 Buffer Object Control

Object	Keyword	Logical Name
Data pages	PAGE	RDM$BIND_PAGE_BUFOBJ_ENABLED
AIJ output	AIJ	RDM$BIND_AIJ_BUFOBJ_ENABLED
RUJ	RUJ	RDM$BIND_RUJ_BUFOBJ_ENABLED
Root file	ROOT	RDM$BIND_ROOT_BUFOBJ_ENABLED

Note

If a logical is defined as "1" then the corresponding buffer will be created as an OpenVMS buffer object.

The "RMU /SET BUFFER_OBJECT" command requires exclusive database access.

The following example demonstrates enabling ROOT buffer objects and disabling PAGE buffer objects. The RMU /DUMP /HEADER command is used to validate the change.

```
$RMU /SET BUFFER_OBJECT /ENABLE=(ROOT) /DISABLE=(PAGE) MF_PERSONNEL
%RMU-I-MODIFIED, Buffer objects state modified
%RMU-W-DOFULLBCK, full database backup should be done to ensure future recovery
$ RMU/DUMP/HEAD MF_PERSONNEL
    .
    .
    .
      - OpenVMS Alpha Buffer Objects are enabled for
```

```
        Root I/O Buffers
    .
    .
    .
```

Prior to Oracle Rdb 7.1, Fast I/O could not be used if the Oracle Rdb global buffers feature was enabled. This restriction has been lifted. When the Very Large Memory (VLM) feature is not being used, buffer objects may be enabled for global buffers.

SQL syntax for controlling these features on a database−wide basis is planned for a future release. As always, care should be taken when utilizing the Fast I/O feature. Buffer objects are memory resident and thus will reduce the amount of physical memory available to OpenVMS for other uses. Buffer object use requires that the user be granted the VMS$BUFFER_OBJECT_USER rights identifier. The system parameter MAXBOBMEM needs to be large enough to allow all buffer objects for all users to be created. For further information regarding Fast I/O, consult the OpenVMS documentation.

5.1.2 RMU Support Added for New OpenVMS Tape Density Values

Oracle Rdb RMU now supports the new OpenVMS tape density and compression values introduced in OpenVMS V7.2−1. The values that can be specified are the same values as those documented by OpenVMS for the VMS INITIALIZE and MOUNT commands as well as other VMS commands that allow tape density and compression to be specified. The existing tape density values supported by the /DENSITY qualifier can continue to be specified for versions of OpenVMS prior to OpenVMS V7.2−1, for OpenVMS tape device drivers that have not been enhanced to use these new density values, and even for OpenVMS tape drivers that have been enhanced to use the new density values. However, if possible, the new density values should be specified for OpenVMS tape device drivers that accept the new density values since in some cases, especially for newer tape drives and tape cartridges, the existing density values may not work as expected. This affects all RMU commands that support the /DENSITY qualifier: RMU/BACKUP, RMU/BACKUP/AFTER_JOURNAL and RMU/OPTIMIZE_AIJ. The new OpenVMS tape density and compression values are sometimes referred to as "MTD" values (multiple tape density) or "MT3" (they translate to internal VMS values that start with "MT3$K_" while the existing density values translate to internal values that start with "MT$K_").

If the existing RMU tape density values are specified for OpenVMS tape device drivers that support the new density values, they will be translated to the new density values if possible; otherwise a warning message will be issued and the existing tape density values will be used since the OpenVMS tape driver that supports the new density values should accept the existing density values in most cases. Similarly, if the new tape density values are specified for VMS tape device drivers that do not support the new density values they will be translated to the existing density values if possible; otherwise a warning message will be issued and the new density value will be translated to the existing "DEFAULT" internal density value (MT$K_DEFAULT) since the tape device driver does not support the new density values. RMU queries the tape device driver at the start of the tape operation to determine if it supports the new density/compression values. If a density related error such as:

```
%RMU−E−DENSITY, TAPE_DEVICE:[000000]DATABASE.BCK; does not support specified
density
```

or

```
%RMU−E−POSITERR, error positioning TAPE_DEVICE:
```

or

```
%RMU-E-BADDENSITY, The specified tape density is invalid for this device
```

is returned, we recommend changing the value specified with the /DENSITY qualifier to one of the new density values for an OpenVMS tape device driver that accepts the new density values or to one of the existing density values for an OpenVMS tape device driver that accepts the existing density values. Generally, it is best to specify the new density values for tape device drivers that accept the new density values and the existing density values for tape device drivers that accept the existing density values to be certain of achieving the desired tape density and compression. The warning message output if an existing density value cannot be translated to one of the new density values is:

```
%RMU_W_MTDSUPPORT, The specified density cannot be translated to an equivalent
 multiple tape density value
```

The warning message output if a new density value cannot be translated to one of the existing density values and is translated to the "DEFAULT" density value is:

```
%RMU-W-NOMTDSUPPORT, The specified multiple tape density cannot be translated
to an equivalent tape density value
```

The default behavior if the /DENSITY qualifier is not specified is to use the current tape density the tape has been set to by an OpenVMS command such as MOUNT or INITIALIZE.

The existing syntax can continue to be used for the existing density values.

```
/DENSITY = density_value
```

where density_value can be one of the following numeric values:

```
0
1
2
800
833
1250
1600
6250
10000
10625
39782
39872
40000
70000
79564
79744
80000
160000
```

For the existing values, compression is determined by the density value and is not specified. For the value to be used for a particular tape drive and tape cartridge, we refer you to the OpenVMS documentation.

For the new values, the syntax to be used is:

```
/DENSITY = new_density_value
```

where new_density_value can be one of the following values:

```
DEFAULT
800
833
1600
6250
3480
3490E
TK50
TK70
TK85
TK86
TK87
TK88
TK89
QIC
8200
8500
8900
DLT8000
SDLT
DDS1
DDS2
DDS3
DDS4
AIT1
AIT2
AIT3
AIT4
COMPACTION
NOCOMPACTION
```

If the new density values and the existing density values are the same (800,833,1600,6250), the intended value will be interpreted as a new value if the tape device driver accepts the new values and as an existing value if the tape device driver only accepts existing values.

For the new values which accept tape compression, the following syntax can be used:

```
/DENSITY = (new_density_value,[NO]COMPACTION)
```

To be used with the second "COMPACTION" parameter, the new density value must be one of the following new density values which accepts compression:

```
DEFAULT
3480
3490E
8200
8500
8900
TK87
TK88
TK89
DLT8000
SDLT
AIT1
AIT2
AIT3
AIT4
```

```
DDS1
DDS2
DDS3
DDS4
```

For the value to be used for a particular tape drive and cartridge, we refer you to the OpenVMS documentation.

USAGE NOTES

- If a density value is desired that is not supported by this syntax, use the VMS INITIALIZE and MOUNT commands to set the tape density and do not specify the /DENSITY qualifier.
- Please refer to the COMPAQ OpenVMS documentation for detailed information on these density values and the tape drives and tape cartridges they should be used with.
- The same density syntax used on the command line can be specified in the PLAN file for PARALLEL RMU backup to tape.

EXAMPLES

The following example uses an existing density value.

```
$ RMU/BACKUP/DENSITY=1250/REWIND/LABEL=(LABEL1,LABEL2) MF_PERSONNEL -
TAPE1:MFP.BCK, TAPE2:
```

The following example uses a new density value with no compression.

```
$ RMU/BACKUP/DENSITY=TK89/REWIND/LABEL=(LABEL1,LABEL2) MF_PERSONNEL -
TAPE1:MFP.BCK, TAPE2:
```

The following example uses the same density value as above but calls for compression.

```
$ RMU/BACKUP/DENSITY=(TK89,COMPACTION)/REWIND/LABEL=(LABEL1,LABEL2) -
MF_PERSONNEL TAPE1:MFP.BCK, TAPE2:
```

5.1.3 Ability to Compress RMU/SHOW STATISTICS Output File Added

A new keyword "COMPRESS" has been added to the list of keywords that can be used with the /OPTIONS qualifier associated with the RMU/SHOW STATISTICS command. Use of this keyword will compress the statistics records written to the output file if a /OUTPUT qualifier is used with the RMU/SHOW STATISTICS command. While replaying the statistics, RMU/SHOW STATISTICS will determine if a record was written using compression or not. If the record was written using compression, it will automatically be decompressed.

If compression is used, the resultant binary file can be read only by RMU/SHOW STATISTICS. The format and contents of a compressed file are not documented or accessible to other applications.

5.1.4 IEEE Floating Point Format for SQL Module Language and Precompiled SQL

Bug 1339112

Support for IEEE floating point formats has been added to SQL Module Language and Precompiled SQL on OpenVMS Alpha platforms (IEEE floating point format support is not available for VAX). There are two IEEE floating point formats: single precision (S−Floating) and double precision (T−Floating). They are 32 and 64 bits in length, respectively. The OpenVMS names for these formats are S_FLOAT and T_FLOAT.

Note that Oracle Rdb always stores floating point numbers internally using the VAX 32−bit and 64−bit types called F−Floating (F_FLOAT) and G−Floating (G_FLOAT), respectively. This means that when IEEE formats are used in a host language program, Oracle Rdb converts back and forth between the VAX and IEEE formats. There are differences in the number of available bits in the fraction and exponent between these formats. Additionally, the IEEE formats have certain exponent values reserved for infinity values. These differences can cause floating point overflow or underflow as well as rounding errors during the conversion process. See Appendix A of the Compaq Portable Mathematics Library in the OpenVMS Operating System documentation for data on the maximum and minimum values for VAX versus IEEE floating point formats.

A new qualifier has been added to the SQL$MOD and SQL$PRE commands which allows host language IEEE 32−bit and 64−bit floating point variables to be used as host variables and parameters in programs calling SQL Module Language procedures and/or containing precompiled SQL (including Dynamic SQL). The format of the qualifier is as follows:

```
/FLOAT={D_FLOAT or G_FLOAT or IEEE_FLOAT}
```

The existing /[NO]G_FLOAT qualifier can continue to be used. The /G_FLOAT qualifier is equivalent to /FLOAT=G_FLOAT and the /NOG_FLOAT qualifier is equivalent to /FLOAT=D_FLOAT.

The meaning of the /FLOAT and /[NO]G_FLOAT qualifiers with SQL$MOD and SQL$PRE closely parallels that of the corresponding qualifiers for the language compilers wherever possible. The details of support vary by language as described below.

5.1.4.1 SQL Module Language (SQL$MOD)

The /FLOAT and /[NO]G_FLOAT qualifiers determine the conversion that SQL Module Language performs on SQL Module Language procedure parameters declared as single or double precision floating point SQL datatypes. SQL floating point datatypes are FLOAT(n), REAL, and DOUBLE PRECISION. See Section 2.3 of the SQL Reference Manual for details. Internally to Rdb, single precision floating point types are represented as F−Floating while double precision floating point types are represented as G−Floating. See Table 3.2 in Section 3.4 of the SQL Reference Manual for more details.

By default, parameters declared as single or double precision floating point type are expected to be passed by the calling host language program in F−Floating and G−Floating format, respectively. This is equivalent to using a qualifier of /FLOAT=G_FLOAT or /G_FLOAT with the SQL$MOD command.

If the command line for SQL$MOD has /FLOAT=D_FLOAT (or /NOG_FLOAT), then the single and double precision floating point parameters are expected to be in F−Floating and D−Floating format respectively. SQL Module Language will convert the double precision parameters between D−Floating and G−Floating formats for both input and output.

If the command line for SQL$MOD has /FLOAT=IEEE_FLOAT, the single and double precision floating point parameters are expected to be in IEEE S−Floating and IEEE T−Floating format, respectively. SQL Module Language will convert between these formats and the internal F−Floating and G−Floating formats for both input and output.

If a parameter of a SQL Module Language procedure is of a record type, any fields of the record which are of floating point types follow the same rules as described above.

In the discussion of actual parameter types below, examples will refer to the following SQL Module Language procedure which is assumed to yield a singleton select:

```
PROCEDURE GET_FLOATS (SQLCODE, REAL :P_FLOAT1, DOUBLE PRECISION :P_FLOAT2);
BEGIN
    SELECT MY_FLOAT1, MY_FLOAT2 INTO :P_FLOAT1, :P_FLOAT2 FROM A_TABLE
    WHERE KEY_VALUE = "1";
END;
```

The floating point formats of the host language program actual parameters must agree with the format expected by the SQL Module Language actual parameter. (See Section 3.4 of the SQL Reference Manual for information concerning actual and formal parameter agreement.)

The host language floating point formats are determined as follows.

Ada

The Ada compiler does not have a /FLOAT or /[NO]G_FLOAT qualifier. The formats of floating point data elements are determined by the declaration of the variable used in the actual parameter. The STANDARD package contains floating point datatypes the format of which is determined by the pragmas FLOAT_REPRESENTATION and LONG_FLOAT. The SYSTEM package contains floating point types which explicitly specify the floating point format associated with the type. These host variable formats and equivalent SQL Module language declarations are detailed in the following table:

Table 5−2 Ada Declarations and Floating Point Formats

Ada Declaration	Compatible SQL$MOD Declaration
pragma FLOAT_REPRESENTATION VAX_FLOAT	SQL$MOD/ADA/FLOAT=G_FLOAT − or −
	SQL$MOD/ADA/FLOAT=D_FLOAT
...	...
FLOAT1 : STANDARD.FLOAT;	REAL :P_FLOAT1 − or −
	FLOAT(24) :P_FLOAT1
pragma FLOAT_REPRESENTATION VAX_FLOAT	SQL$MOD/ADA/FLOAT=G_FLOAT
pragma LONG_FLOAT G_FLOAT	...
...	
FLOAT1 : STANDARD.LONG_FLOAT;	DOUBLE_PRECISION :P_FLOAT1 − or −
	FLOAT(53) :P_FLOAT1
pragma FLOAT_REPRESENTATION VAX_FLOAT	SQL$MOD/ADA/FLOAT=D_FLOAT
pragma LONG_FLOAT D_FLOAT	...
...	

FLOAT2 : STANDARD.LONG_FLOAT;	DOUBLE_PRECISION :P_FLOAT2 − or −
	FLOAT(53) :P_FLOAT2
...	SQL$MOD/ADA/FLOAT=D_FLOAT
	...
FLOAT1 : SYSTEM.F_FLOAT;	REAL :P_FLOAT1 − or −
	FLOAT(24) :P_FLOAT1
FLOAT2 : SYSTEM.D_FLOAT;	DOUBLE_PRECISION :P_FLOAT2 − or −
	FLOAT(53) :P_FLOAT2
...	SQL$MOD/ADA/FLOAT=G_FLOAT
	...
FLOAT1 : SYSTEM.F_FLOAT;	REAL :P_FLOAT1 − or −
	FLOAT(24) :P_FLOAT1
FLOAT2 : SYSTEM.G_FLOAT;	DOUBLE_PRECISION :P_FLOAT2 − or −
	FLOAT(53) :P_FLOAT2
...	SQL$MOD/ADA/FLOAT=IEEE_FLOAT
	...
FLOAT1 : SYSTEM.IEEE_SINGLE_FLOAT;	REAL :P_FLOAT1 − or −
	FLOAT(24) :P_FLOAT1
FLOAT2 : SYSTEM.IEEE_DOUBLE_FLOAT;	DOUBLE_PRECISION :P_FLOAT2 − or −
	FLOAT(53) :P_FLOAT2

See Table 3.3 in Section 3.4 of the SQL Reference Manual for more information on equivalency between SQL and Ada data types.

The following example shows an Ada code fragment which is compatible with the GET_FLOATS sample procedure when the SQL Module Language program has been compiled with /FLOAT=IEEE_FLOAT:

```
procedure GET_FLOATS (
    SQLCODE : out INTEGER;
    P_FLOAT1 : out IEEE_SINGLE_FLOAT;
    P_FLOAT2: out IEEE_DOUBLE_FLOAT
    );
pragma INTERFACE (NONADA, GET_FLOATS);
SQLCODE : INTEGER;
FLOAT1 : SYSTEM.IEEE_SINGLE_FLOAT;
FLOAT2 : SYSTEM.IEEE_DOUBLE_FLOAT;
...
GET_FLOATS( SQLCODE, FLOAT1, FLOAT2);
```

BASIC

BASIC provides a /REAL_SIZE qualifier which can be used to specify not only the size but the format of floating point variables declared using the REAL keyword. The relevant values for this qualifier for IEEE floating point formats are SFLOAT and TFLOAT. These values specify that REAL variables are to be of type S−Floating or T−Floating, respectively. BASIC also provides the OPTION command which allows the size and format of a REAL to be specified in a more local scope.

Additionally, BASIC has native datatypes (SFLOAT and TFLOAT) which explicitly specify S−Floating and

T−Floating variables, respectively. See Table 3.4 in Section 3.4 of the SQL Reference Manual for more information on equivalency between SQL and BASIC data types.

The following example shows a BASIC code fragment which is compatible with the GET_FLOATS sample procedure:

```
EXTERNAL GET_FLOATS(LONG, SFLOAT, TFLOAT)
DECLARE LONG SQLCODE
DECLARE SFLOAT FLOAT1
DECLARE TFLOAT FLOAT2
...
CALL GET_FLOATS(SQLCODE, FLOAT1, FLOAT2)
```

C

C has /FLOAT and /[NO]G_FLOAT qualifiers which work identically to those for SQL$MOD (except the default may be different). That is, the format of the floating point variables in the C program is determined by the qualifier. C has native types of "float" and "double" which are 32−bit and 64−bit floating point numbers, respectively. See Table 3.5 in Section 3.4 of the SQL Reference Manual for more information on equivalency between SQL and C data types.

The following example shows a C code fragment which is compatible with the GET_FLOATS sample procedure provided that both the C module and the SQL Module Language program were compiled with the same setting of the /FLOAT or /[NO]G_FLOAT qualifier:

```
extern void GET_FLOATS (
    long *SQLCODE,
    float *P_FLOAT1,
    double *P_FLOAT2
);
long SQLCODE;
float float1;
double float2;
...
GET_FLOATS( &SQLCODE, &float1, &float2);
```

COBOL

On the Alpha platform, COBOL has a /FLOAT qualifier with the same options as SQL$MOD (except the default is D_FLOAT). There is no /[NO]G_FLOAT qualifier for COBOL. The /FLOAT qualifier works identically to that of SQL$MOD. That is, the format of the floating point variables in the COBOL program is determined by the qualifier. COBOL has native types of COMP−1 and COMP−2 which are 32−bit and 64−bit floating point numbers, respectively. See Table 3.6 in Section 3.4 of the SQL Reference Manual for more information on equivalency between SQL and COBOL data types.

The following example shows a COBOL code fragment which is compatible with the GET_FLOATS sample procedure provided that both the COBOL program and the SQL Module Language program were compiled with the same setting of the /FLOAT qualifier:

```
DATA DIVISION.
WORKING−STORAGE SECTION.
01      SQLCODE PIC S9(9)   USAGE COMP.
01      FLOAT1  COMP−1.
01      FLOAT2  COMP−2.
...
```

```
CALL "GET_FLOATS" USING SQLCODE, FLOAT1, FLOAT2.
```

FORTRAN

FORTRAN has /FLOAT and /[NO]G_FLOAT qualifiers which work identically to those for SQL$MOD (except the default may be different). That is, the format of the floating point variables in the FORTRAN program is determined by the qualifier. FORTRAN has native types of "real" and "real*4" which are 32−bit floating point numbers and "double precision" and "real*8" which are 64−bit floating point numbers. See Table 3.7 in Section 3.4 of the SQL Reference Manual for more information on equivalency between SQL and FORTRAN data types.

The following example shows a FORTRAN code fragment which is compatible with the GET_FLOATS sample procedure provided that both the FORTRAN module and the SQL Module Language program were compiled with the same setting of the /FLOAT or /[NO]G_FLOAT qualifier:

```
integer*4 SQLCODE
real*4 float1
real*8 float2
...
CALL GET_FLOATS( SQLCODE, float1, float2)
```

Pascal

Pascal has /FLOAT and /[NO]G_FLOAT qualifiers which work similarly to those for SQL$MOD (except the default may be different). That is, the format of floating point variables of certain data types in the Pascal program is determined by the qualifier. The Pascal native data types affected by the qualifiers are REAL, SINGLE and DOUBLE. The first two of these are 32−bit floating point numbers and the final one is a 64−bit floating point number. Pascal also has an attribute called FLOAT which can be used to affect the format of floating point similarly to the /FLOAT qualifier but with a more local scope. Specifically, IEEE floating point format can be specified by using the IEEE_FLOAT keyword with the FLOAT Attribute.

In addition, Pascal has several format−specific floating point data types which specify a particular format regardless of the qualifier settings. The format−specific data types include S_FLOAT and T_FLOAT which are IEEE 32−bit and 64−bit floating point numbers, respectively. (Note that this change also includes support for the Pascal format−specific datatypes of F_FLOAT, D_FLOAT, and G_FLOAT.) See Table 3.8 in Section 3.4 of the SQL Reference Manual for more information on equivalency between SQL and Pascal data types.

The following example shows a Pascal code fragment which is compatible with the GET_FLOATS sample procedure when the SQL Module Language program was compiled with /FLOAT=IEEE_FLOATING:

```
sqlcode : INTEGER;
float1 : S_FLOAT;
float2 : T_FLOAT:
PROCEDURE GET_FLOATS
(   VAR SQLCODE     : INTEGER;
    VAR FLOAT_1     : S_FLOAT;
    VAR FLOAT_2     : T_FLOAT );
EXTERNAL;
...
GET_FLOATS( sqlcode, float1, float2)
```

PL/I

PL/I has /FLOAT and /[NO]G_FLOAT qualifiers which work identically to those for SQL$MOD (except the default may be different). That is, the format of the floating point variables in the PL/I program is determined by the qualifier. PL/I has a native type of FLOAT which can be a 32−bit or 64−bit floating point number depending on the size specification. See Table 3.9 in Section 3.4 of the SQL Reference Manual for more information on equivalency between SQL and PL/I types.

The following example shows a PL/I code fragment which is compatible with the GET_FLOATS sample procedure provided that both the PL/I module and the SQL Module Language program were compiled with the same setting of the /FLOAT or /[NO]G_FLOAT qualifier:

```
DECLARE GET_FLOATS EXTERNAL ENTRY (
        ANY REFERENCE, ANY REFERENCE, ANY REFERENCE);
DECLARE SFLOAT FLOAT(24) BINARY,
        TFLOAT FLOAT(53) BINARY,
        SQLCODE BIN FIXED(31);
...
CALL GET_FLOATS( SQLCODE, SFLOAT, TFLOAT );
```

5.1.4.2 Precompiled SQL (SQL$PRE)

The SQL Precompiler translates embedded SQL into host language declarations and procedure calls. In addition, it generates the procedures behind the procedure calls. The /FLOAT and /[NO]G_FLOAT qualifiers for SQL$PRE determine the floating point format that SQL$PRE assumes for host language variables and, hence, determines the conversions that will be made internal to the generated SQL procedures. When SQL$PRE calls the host language compiler to process the precompiled program, it passes an equivalent qualifier to its /FLOAT or /[NO]G_FLOAT qualifier where supported by the host language. This means that, to the extent that the floating point format of host language variables is determined by a /FLOAT or /[NO]G_FLOAT qualifier, the floating point formats of the host language variables and the parameters of procedure calls generated by SQL$PRE are guaranteed to be compatible. When the host language provides a type which explicitly declares the floating point format of the an individual variable, SQL$PRE uses that information to determine the conversion needed regardless of the setting of the /FLOAT or /[NO]G_FLOAT qualifier.

The SQL Precompiler's default floating point format for single or double precision floating point types is F−Floating and G−Floating format, respectively. This is equivalent to using a qualifier of /FLOAT=G_FLOAT or /G_FLOAT with the SQL$PRE command.

If a parameter of a SQL Module Language procedure is of a record type, any fields of the record which are of floating point types follow the same rules as described above.

There are a few cases where a host language provides mechanisms for specifying floating point format which are not recognized by SQL$PRE. In these cases, it is the developer's responsibility to ensure that the format is what SQL$PRE expects. These cases are described in the host language−specific sections that follow. In these sections, selects will be shown from a table defined as follows:

```
CREATE TABLE TESTTBL (
    KEYFIELD CHAR(10) PRIMARY KEY,
    FLOAT1   REAL,
    FLOAT2   DOUBLE PRECISION);
```

Ada

Refer to Section 4.5.2 of the SQL Language Reference Manual for information about supported Ada floating point variable declarations. SQL$PRE now supports the format−explicit types IEEE_SINGLE_FLOAT and IEEE_DOUBLE_FLOAT in package SYSTEM in addition to the package SYSTEM floating point types documented in the SQL Reference Manual. These newly supported types correspond to 32−bit and 64−bit IEEE floating point numbers, respectively.

In addition, the Ada pragma FLOAT REPRESENTATION can be set to IEEE_FLOAT to override the default formats of the intrinsic Ada type FLOAT as well as the floating point types in packages STANDARD and SQL_STANDARD. If IEEE floating point format is specified using the pragma, a /FLOAT=IEEE_FLOATING qualifier is required for the SQL$PRE command.

Note: SQL$PRE will issue a warning (%SQL−W−NOFLOAT) if you use a /FLOAT qualifier with an /ADA qualifier because the Ada command does not have a /FLOAT qualifier. But if you use a pragma FLOAT REPRESENTATION to override the default floating point formats, you must use the /FLOAT qualifier to let SQL$PRE know about this floating point format since it does not recognize the pragma. Simply ignore the warning. In addition to supporting IEEE formats, SQL$PRE will now allow the default G_FLOAT format for 64−bit floating point types to be overridden using a combination of the pragma FLOAT REPRESENTATION specifying VAX_FLOAT and the pragma LONG FLOAT specifying D_FLOAT. To use this combination, specify a SQL$PRE qualifier of /FLOAT=D_FLOAT.

The following example shows an Ada program with embedded SQL that will work correctly with SQL$PRE/ADA/FLOAT=IEEE:

```
PRAGMA FLOAT REPRESENTATION IEEE_FLOAT;
WITH SYSTEM; USE SYSTEM;
WITH STANDARD; USE STANDARD;
WITH SQL_STANDARD; USE SQL_STANDARD;
...
PROCEDURE TESTIT IS
EXEC SQL BEGIN DECLARE SECTION;
KEYFIELD   : STRING(1..10);
FLOATER    : LONG_FLOAT;           -- package STANDARD
SQLFLOATER : REAL;                 -- package SQL_STANDARD
GFLOATER   : G_FLOAT;              -- package SYSTEM
SFLOATER   : IEEE_SINGLE_FLOAT;    -- package SYSTEM
TFLOATER   : IEEE_DOUBLE_FLOAT;    -- package SYSTEM
EXEC SQL END DECLARE SECTION;
...
BEGIN
...
KEYFIELD := "1.0       ";
EXEC SQL SELECT FLOAT1, FLOAT2 INTO :SQLFLOATER, :GFLOATER
    WHERE KEYFIELD = :KEYFIELD;
...
KEYFIELD := "2.0       ";
EXEC SQL SELECT FLOAT1, FLOAT2 INTO :SFLOATER, :TFLOATER
    WHERE KEYFIELD = "KEYFIELD;
...
KEYFIELD := "3.0       ";
EXEC SQL SELECT FLOAT1, FLOAT2 INTO :FLOATER, TFLOATER
    WHERE KEYFIELD = KEYFIELD;
```

BASIC

The SQL Precompiler does not support BASIC.

C, COBOL, FORTRAN, PL/I

The compilers for these languages have /FLOAT and /[NO]G_FLOAT qualifiers which are totally analogous to those of SQL$PRE. Consequently, programs which contain embedded SQL can simply be recompiled using /FLOAT=IEEE_FLOATING and will link and run with other object modules which have been compiled with /FLOAT=IEEE_FLOATING. Refer to Section 4.5.3 of the SQL Language Reference Manual for information about supported C floating point variable declarations. Refer to Section 4.5.4 of the SQL Language Reference Manual for information about supported COBOL floating point variable declarations. Refer to Section 4.5.5 of the SQL Language Reference Manual for information about supported FORTRAN floating point variable declarations. Refer to Section 4.5.7 of the SQL Language Reference Manual for information about supported PL/I floating point variable declarations.

Pascal

The Pascal compiler has /FLOAT and /[NO]G_FLOAT qualifiers which are totally analogous to those of SQL$PRE. The qualifiers affect all the Pascal floating point datatypes which don't explicitly imply a floating point qualifier. These Pascal datatypes are REAL, SINGLE, and DOUBLE. Refer to Section 4.5.6 of the SQL Language Reference Manual for information about supported Pascal floating point datatypes. Programs using these datatypes can simply be recompiled with /FLOAT=IEEE_FLOATING and will link and run with other object modules which have been compiled with /FLOAT=IEEE_FLOATING. (Note however that SQL$PRE does not support the FLOAT attribute.)

In addition, SQL$PRE now supports the Pascal floating point datatypes which explicitly specify the floating point format. These newly supported types are F_FLOAT, D_FLOAT, G_FLOAT, S_FLOAT, and T_FLOAT. The first three use the VAX formats with the same name while the last two are IEEE 32−bit and 64−bit floating point formats, respectively. When SQL$PRE compiles programs that contain variables declared with any of these datatypes, it provides the appropriate conversion regardless of the value of the /FLOAT qualifier.

The following example illustrates a Pascal program using variables with various floating point formats. In this example, the value of the /FLOAT qualifier is not important internally to the program and is only relevant if the resulting module must pass floating point parameters to or from some external module.

```
PROGRAM TEST_PASCAL (INPUT,OUTPUT)
EXEC SQL INCLUDE SQLCA;
VAR
KEYFIELD   : PACKED ARRAY [1..10] OF CHAR;
PAS_TFLOAT : T_FLOAT;
PAS_SFLOAT : S_FLOAT;
PAS_REAL   : REAL;
PAS_DOUBLE : DOUBLE;
PAS_GFLOAT : G_FLOAT;
PAS_FFLOAT : F_FLOAT;
PAS_DFLOAT : D_FLOAT;
BEGIN
...
KEYFIELD := '1.0';
EXEC SQL SELECT FLOAT1, FLOAT2 INTO :PAS_SFLOAT, :PAS_TFLOAT
    WHERE KEYFIELD = :KEYFIELD;
...
KEYFIELD := '2.0';
EXEC SQL SELECT FLOAT1, FLOAT2 INTO :PAS_REAL, :PAS_GFLOAT
    WHERE KEYFIELD = :KEYFIELD;
...
KEYFIELD := '3.0';
```

```
EXEC SQL SELECT FLOAT1, FLOAT2 INTO :PAS_FFLOAT, :PAS_DOUBLE
    WHERE KEYFIELD = :KEYFIELD;
...
KEYFIELD := '4.0';
EXEC SQL SELECT FLOAT1, FLOAT2 INTO :PAS_DFLOAT, :PAS_TFLOAT
    WHERE KEYFIELD = :KEYFIELD;
...
```

5.1.4.3 Use of the Dynamic Descriptor Areas (SQLDA and SQLDA2)

Dynamic SQL can now pass floating point parameters in IEEE formats using the SQLDA and SQLDA2. See Appendix D of the SQL Reference Manual for information on using the SQLDA including which languages support it. The floating point format that SQL*PRE assumes is determined by the value of the /FLOAT or /[NO]G_FLOAT qualifier. It is the developer's responsibility to provide a pointer to a variable of the appropriate type according to the rules in the sections above. The following example illustrates using SQLDA with an Ada program processed by the SQL Precompiler with /FLOAT=IEEE.

```
PRAGMA FLOAT REPRESENTATION IEEE_FLOAT;
WITH SYSTEM; USE SYSTEM;
WITH STANDARD; USE STANDARD;
...
PROCEDURE TESTIT IS
EXEC SQL BEGIN DECLARE SECTION;
FLOATER    : FLOAT;                -- package STANDARD
TFLOATER   : IEEE_DOUBLE_FLOAT;    -- package SYSTEM
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLDA;
...
BEGIN
SQLDA := NEW SQLDA_RECORD;
SQLDA.SQLN := 255;
...
EXEC SQL PREPARE the_stmt FROM
    'select float1, float2 from testtbl where keyfield = ''10000''';
EXEC SQL DESCRIBE the_stmt SELECT LIST INTO SQLDA;
SQLDA.SQLVAR(1).SQLDATA := FLOATER'ADDRESS;
SQLDA.SQLVAR(2).SQLDATA := TFLOATER'ADDRESS;
EXEC SQL DECLARE the_cursor CURSOR FOR the_stmt;
EXEC SQL OPEN the_cursor;
EXEC SQL FETCH the_cursor USING DESCRIPTOR sqlda;
EXEC SQL CLOSE the_cursor;
```

5.1.4.4 Use of Common Data Dictionary (CDD)

Both SQL Module Language and Precompiled SQL allow field and record definitions to be imported from a CDD repository. CDD provides various floating point datatypes which explicitly specify the various VAX floating point formats. However no IEEE floating point format datatypes are provided. Consequently, the ability to use CDD fields of floating point types and CDD records containing floating point fields is very limited when IEEE floating point formats are used.

When using a /FLOAT=IEEE_FLOATING qualifier, CDD records and fields with floating point types can only be included with the combination of C and embedded SQL. In this case, the floating point format specified in the CDD repository definition is simply ignored by both the precompiler and the C compiler and

the floating point format is determined by the value of the /FLOAT or /[NO]G_FLOAT qualifier.

5.1.5 INCLUDE_DB_NAME Event Attribute for RMU/SHOW STATISTICS User Defined Events

Bug 2156905

RMU/SHOW STATISTICS user defined events gives you the ability to have a user defined command procedure/program be invoked when the event is triggered. The parameters used for invoking this procedure/program contain certain information about the event. The parameter P2 holds the name of the statistic on which the event is based.

If a user is monitoring several databases for the same statistic and has events defined for all the databases, the user would not know on which database the event got triggered. To address this problem, the INCLUDE_DB_NAME attribute has been added. If this attribute is set to "1", the parameter P2 will hold both statistic name and database name. It will be of the form "statistics name" for "database name". If the database name is not needed, the INCLUDE_DB_NAME attribute can be set to "0" or not be included in the event description. By default, INCLUDE_DB_NAME has a value of "0".

The following are examples of events defined with the INCLUDE_DB_NAME attribute.

```
EVENT_DESCRIPTION="ENABLE 'transactions' \
MAX_CUR_RATE \
INITIAL 3 \
EVERY 1 \
LIMIT 0 \
INCLUDE_DB_NAME 1 \
INVOKE DB";


EVENT_DESCRIPTION="ENABLE 'transactions' \
MAX_CUR_RATE \
INITIAL 3 \
EVERY 1 \
LIMIT 0 \
INCLUDE_DB_NAME 0 \
INVOKE DB";
```

The INCLUDE_DB_NAME attribute is available in Oracle Rdb Release 7.1.0.2.

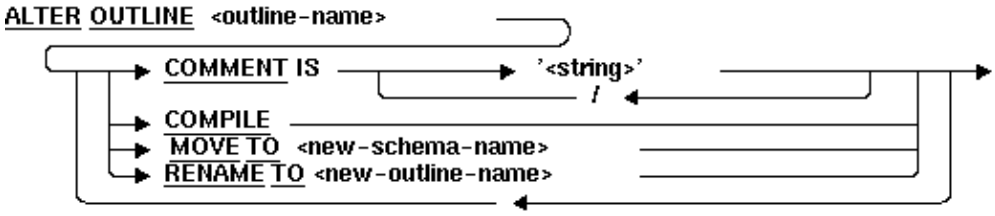
5.1.6 New ALTER OUTLINE Statement

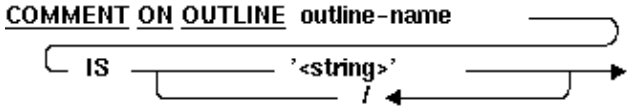
This release of Oracle Rdb 7.1 includes an ALTER OUTLINE and a COMMENT ON OUTLINE statement.

You can use the ALTER OUTLINE statement:

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

FORMAT





USAGE NOTES

- The outline name can be prefixed with a database alias name. For example:

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

  In a multischema database, the name can also include a schema name and catalog name.
- The COMPILE option can be applied to query outlines that have been made invalid by DROP TABLE or DROP INDEX. If the tables and indices have been recreated then the query outline will be made valid again (i.e. once re−validated the optimizer will try to use that query outline).
  Note: there is a possibility that the query outline, although marked valid, will not be used because of changes in the index definition. There is too little information stored with the query outline to perform a complete consistency check. If possible, queries using this outline should be run to verify correct index and table usage.
  If the query outline is currently valid then this clause is ignored by Rdb.
- MOVE TO is valid only for multischema databases. You must be attached explicitly or implicitly with the MULTISCHEMA IS ON clause. The MOVE TO clause can be used to move the query outline to a different catalog and schema. An error will be raised if this clause is specified in a non−multischema environment.
  The target catalog and schema must exist in this database.
- The RENAME TO clause can be used to change the name of the outline. The new name must not already exist in the database.
  If RENAME TO is used in a multischema database attached with MULTISCHEMA IS ON, then only the multischema name is modified not the STORED NAME of the object. To change the STORED NAME of the query outline, you must attach to the database explicitly with the MULTISCHEMA IS OFF clause (see the example below). Please note that the STORED NAME for the query outline may have been generated by Rdb.
  Note that any queries using the OPTIMIZE USING clause will also need to be changed to reference this new outline name.
- The COMMENT IS clause can be used to modify the comment stored with the query outline. The COMMENT ON statement is identical in function to the ALTER OUTLINE ... COMMENT IS clause.

EXAMPLES

Example 1: Changing the comment on a query outline.

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

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

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

Example 3: Changing the name of a query outline.

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

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

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

```
SQL> set flags 'strategy';
```

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

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

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

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

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

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

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

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

5.1.7 DROP Statement Now Includes IF EXISTS Clause

The following DROP statements now support a new IF EXISTS option which allows the DROP to succeed even if the named object is not in the database.

- DROP CATALOG
- DROP COLLATING SEQUENCE
- DROP CONSTRAINT
- DROP DOMAIN
- DROP FUNCTION
- DROP INDEX
- DROP MODULE
- DROP OUTLINE
- DROP PROCEDURE
- DROP PROFILE
- DROP SEQUENCE
- DROP SCHEMA
- DROP STORAGE MAP
- DROP SYNONYM
- DROP TABLE
- DROP TRIGGER
- DROP USER
- DROP ROLE
- DROP VIEW

Usage Notes

- No error is reported if the referenced object does not exist in the database. Use IF EXISTS in SQL command scripts to avoid unwanted error messages.
- For multischema databases, the IF EXISTS clause may not operate as expected because the object is internally deleted using the STORED NAME which may be different from that specified by the DROP statement. Currently, the IF EXISTS clause assumes that the multischema name and the stored name are identical.

Example: Adding New Definitions to a Database

When updating metadata definitions using a predefined SQL script, it is sometimes required to remove objects that may not be present in all databases being maintained. Adding a DROP VIEW, for instance, will result in an error as shown here.

```
SQL> drop view CURRENT_INFO;
%SQL-F-RELNOTDEF, Table CURRENT_INFO is not defined in database or schema
SQL> create view CURRENT_INFO
cont> ...etc...
```

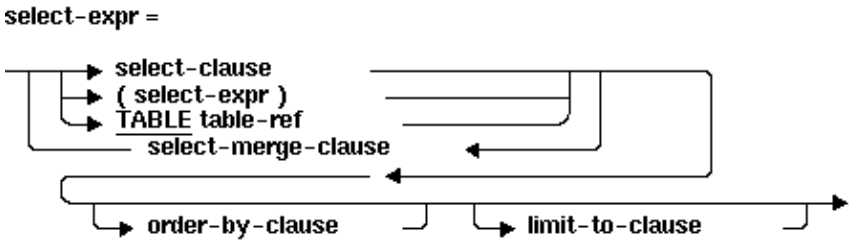
By using the IF EXISTS clause, the error message is supressed and makes for a less confusing execution of the maintance script.

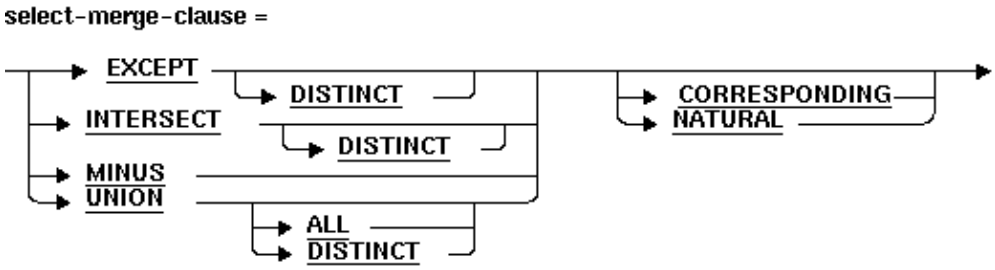
```
SQL> drop view CURRENT_INFO if exists;
SQL> create view CURRENT_INFO
cont> ...etc...
```

5.1.8 New EXCEPT, INTERSECT and MINUS Operators

This release of Oracle Rdb adds three new operators to the select expression syntax. The new operators, EXCEPT, INTERSECT and MINUS, are all forms of select table merge operations.

FORMAT





ARGUMENTS

- EXCEPT
  EXCEPT DISTINCT
  The EXCEPT DISTINCT is used to create a result table from the first select expression except for those row values that also occur in the second select expression.
  DISTINCT is the default so EXCEPT and EXCEPT DISTINCT are identical operations. EXCEPT conforms to the ANSI and ISO SQL:1999 Database Language Standard.

  Note

  EXCEPT is not commutative. That is, A EXCEPT B may result in a different set of rows from B EXCEPT A. This is demonstrated by the examples below.

- INTERSECT
  INTERSECT DISTINCT
  The INTERSECT DISTINCT operator is used to create a result table from the first select expression of those row values that also occur in the second select expression.
  DISTINCT is the default so INTERSECT and INTERSECT DISTINCT are identical operations. INTERSECT conforms to the ANSI and ISO SQL:1999 Database Language Standard.

  Note

  In general INTERSECT is commutative. That is, A INTERSECT B results in the same set of rows from B INTERSECT A. This is demonstrated by the examples below. However, care should be taken when using LIMIT TO within the different

branches of the INTERSECT as this will make the result non−deterministic because of different solution strategies employed by the Rdb optimizer.

- MINUS
  The MINUS operator is a synonym for the EXCEPT DISTINCT operator and is provided for language compatibility with the Oracle RDBMS SQL language.
- UNION
  UNION ALL
  UNION DISTINCT
  Please refer to the existing Rdb documentation for information on the UNION operator.
- CORRESPONDING
  The UNION, EXCEPT, MINUS, and INTERSECT operators can be followed by the keyword CORRESPONDING. This causes the two select lists of the select−merge−clause to be compared by name. Only those column names which appear in both lists are retained for the resulting query table. The name is either the column name or the name provided by the AS clause. If there are no names in common, or a column name appears more than once in a select list, then an error is reported.

USAGE NOTES

- The EXCEPT DISTINCT operator can be rewritten to use the NOT ANY predicate. In fact, the Rdb server currently implements EXCEPT DISTINCT in this way. Consider this example:

```
SQL> select manager_id from departments
cont> except distinct
cont> select employee_id from employees;
```

  This query could be rewritten as:

```
SQL> select manager_id
cont> from departments d
cont> where not exists (select *
cont>                   from employees e
cont>                   where e.employee_id = d.manager_id
cont>                      or (e.employee_id is null
cont>                          and d.manager_id is null));
```

  As you can see, even for this simple query, the EXCEPT format is easier to read. As the number of columns selected increases so does the complexity of the NOT EXISTS subquery.
- The INTERSECT DISTINCT operator can be rewritten to use the EXISTS predicate. In fact, the Rdb server currently implements INTERSECT DISTINCT in this way. Consider this example which displays all managers who are also employees:

```
SQL> select manager_id from departments
cont> intersect distinct
cont> select employee_id from employees;
```

  This query could be rewritten as:

```
SQL> select manager_id
cont> from departments d
```

```
cont> where exists (select *
cont>               from employees e
cont>               where e.employee_id = d.manager_id
cont>                  or (e.employee_id is null
cont>                      and d.manager_id is null));
```

As you can see, even for this simple query, the INTERSECT format is easier to read. As the number of columns selected increases so does the complexity of the EXISTS subquery.

- For both EXCEPT and INTERSECT, all duplicate rows are eliminated. For the purposes of these operators, a row is considered a duplicate if each value in the first select list is equal to the matching column in the second select list, or if both these columns are NULL.
  The duplicate matching semantics can be clearly seen in the rewritten queries which use NOT EXISTS and EXISTS.

EXAMPLES

The following examples show the new clauses in use.

Example 1: Using CORRESPONDING as a Shorthand for the Select List

This example derives results from tables with some column names in common. Here the table RETIRED_EMPLOYEES contains a subset of the columns from EMPLOYEES (EMPLOYEE_ID and LAST_NAME) as well as some new columns to describe the retired employee (e.g. RETIRE_DATE). CORRESPONDING is used to match the common column names and produce this report.

```
SQL> select *, 'retired' as status from RETIRED_EMPLOYEES
cont> union corresponding
cont> select *, 'working' as status from EMPLOYEES e
cont> order by status;
 EMPLOYEE_ID   LAST_NAME        STATUS
 00207         Babbin           retired
 00173         Bartlett         retired
 ...etc...
```

Example 2: Changing a Result Name by Applying the AS Clause

This example shows the use of the AS clause to name the AVG statistical expression the same in each part of the UNION clause. CORRESPONDING will align these two columns. Without the AS clause, these column expressions would have been eliminated from the UNION result table.

```
SQL> select pnum,
cont>       avg(weight) as AVG edit using 'ZZZZ99.99'
cont>  from p
cont>  group by pnum
cont> union corresponding
cont> select pnum,
cont>        avg(qty) as AVG
cont>  from spj
cont>  group by pnum;
 PNUM           AVG
 P1           12.00
 P1          333.33
 P2           17.00
 P2          150.00
 P3           17.00
```

```
 P3          388.89
 P4           14.00
 P4          650.00
 P5           12.00
 P5          450.00
 P6           19.00
 P6          325.00
12 rows selected
SQL>
```

Example 3: EXCEPT DISTINCT Operator

Here we use UNION DISTINCT to derive the full set of EMPLOYEE_ID values. Since all managers are also employees, this list should return the same rows as a query on EMPLOYEES. It is used here to show the differences between these similarly structured operators.

```
SQL> select manager_id from departments
cont> union distinct
cont> select employee_id from employees;
 MANAGER_ID
 00164
 00165
 00166
.
.
.
 00435
 00471
100 rows selected
```

Make sure that all managers are also employees. List all managers who are not employees. The result shows that there are no managers in this list.

```
SQL> select manager_id from departments
cont> except distinct
cont> select employee_id from employees;
0 rows selected
```

List all employees who are not managers. Or, stated in a different way, list all employees, except those that are managers. This is done simply by reversing the order of the select expressions from the previous query. Note that we get quite a different result.

```
SQL> select employee_id from employees
cont> except distinct
cont> select manager_id from departments;
 EMPLOYEE_ID
 00165
 00167
 00169
.
.
.
 00416
 00435
74 rows selected
```

Example 4: INTERSECT DISTINCT Operator

Show the managers who are also employees.

```
SQL> select manager_id from departments
cont> intersect distinct
cont> select employee_id from employees;
 MANAGER_ID
 00164
 00166
 00168
.
.
.
 00418
 00471
26 rows selected
```

INTERSECT DISTINCT is commutative so reversing the select expressions will yield the same result set. However, the same ordering of these rows is not guaranteed unless an ORDER BY clause is applied to the result.

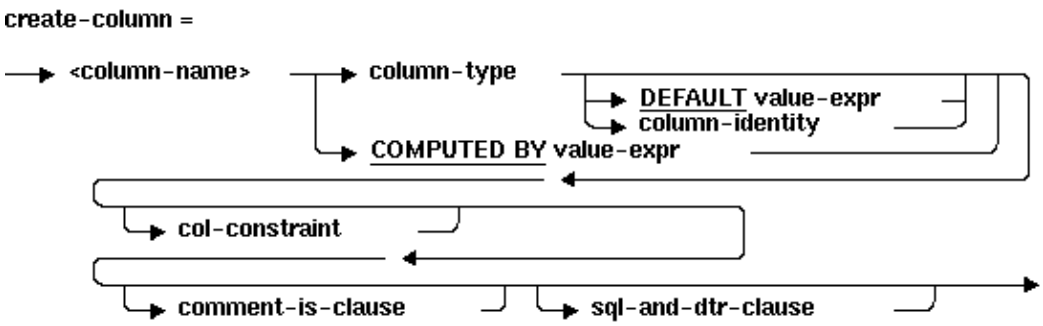
```
SQL> select employee_id from employees
cont> intersect distinct
cont> select manager_id from departments;
 EMPLOYEE_ID
 00164
 00166
 00168
.
.
.
 00418
 00471
26 rows selected
```

5.1.9 IDENTITY Attribute Now Supported by Oracle Rdb

This release of Rdb, Release 7.1.0.2, supports the IDENTITY attribute for a table. This special column attribute is a shorthand mechanism for adding and maintaining a unique id generator for any table. This feature is based on both the SEQUENCE and AUTOMATIC columns feature.

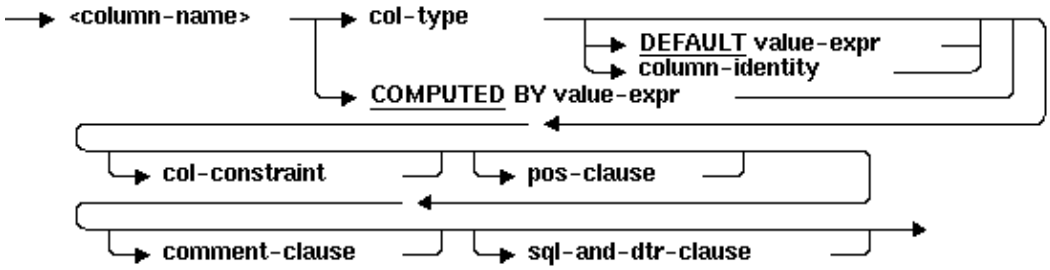
The IDENTITY attribute can be specified by CREATE or ALTER TABLE.

FORMAT



column-identity =

alter-add-column =



USAGE NOTES

- Only columns of the type TINYINT, SMALLINT, INTEGER, or BIGINT can use the IDENTITY attribute. These types must default to or have a zero scale. Domains may be referenced if they have these types.
- This attribute implicitly creates a system sequence with the same name as the table in which it resides. This sequence can be modified using the ALTER SEQUENCE statement, however, the sequence can only be dropped using ALTER TABLE ... DROP COLUMN, or by DROP TABLE. There can only be one column using IDENTITY in any one table.
- The START WITH and INCREMENT BY values for the created sequence default to 1 if omitted from the IDENTITY specification. These values can be provided with the IDENTITY attribute. See the examples below.
- This attribute implicitly changes the column to be an AUTOMATIC INSERT column, therefore it becomes a READ ONLY column. Please refer to the documentation on AUTOMATIC columns for more information.
- If a TRUNCATE TABLE is executed for this table, the special sequence is reset to the initial starting value.
- DEFAULT and IDENTITY may not both be specified for a column.
- AUTOMATIC and IDENTITY may not both be specified for a column.
- When adding an IDENTITY column to an existing table using ALTER TABLE ... ADD COLUMN, an implicit update query is executed on the table and a value is assigned to the identity column for each row. The order of rows updated, and hence the values assigned to each row, is dependent on the query strategy chosen for the update.
- Constraints, especially PRIMARY KEY, can be defined for the identity column.
- Indices can be defined which include the identity column.
- The IDENTITY attribute implicitly creates a sequence with the same name as the table. This sequence name can be used by SHOW SEQUENCE, GRANT and REVOKE, and ALTER SEQUENCE. When granting role and user access to the table, the database administrator will need to also grant SELECT privilege to the sequence.
- The DROP SEQUENCE statement is not supported with an identity derived sequence.
- The CURRVAL pseudo column can be used after an insert has been performed so that the sequence number can be used in related tables. For instance:

```
SQL> insert into ORDER values (...);
SQL> insert into ORDER_LINES (ORDER.CURRVAL, ...);
SQL> insert into ORDER_LINES (ORDER.CURRVAL, ...);
```

This example shows that the FOREIGN KEY value is selected using a reference to the table name followed by the CURRVAL clause.

- The NEXTVAL pseudo column cannot be used to fetch a new identity value. Only an INSERT on the table can generate a new identity value.
- If the INSERT on the table is rolled back or fails due to a constraint or trigger error condition, then the used identity values are discarded. If a row is deleted from the table, the identity value is not reused. For an exception to the reuse rule, see the usage note on TRUNCATE TABLE.

EXAMPLES

Example 1: Using the IDENTITY attribute

This simplified order entry database uses IDENTITY on all tables to generate unique values for the table primary key field.

```
SQL> create domain MONEY as INTEGER (2);
SQL> create domain CUSTOMER_ID as INTEGER;
SQL> create domain PRODUCT_ID as INTEGER;
SQL> create domain ORDER_ID as INTEGER;
SQL> create domain LINE_NUMBER as INTEGER
cont>     check (VALUE > 0 and VALUE IS NOT NULL)
cont>     not deferrable;
SQL>
SQL> create table PRODUCTS
cont>     (product_id           PRODUCT_ID identity primary key,
cont>      product_name         char (100),
cont>      unit_price           MONEY,
cont>      unit_name            char (10)
cont>     );
SQL> create unique index PRODUCTS_IX on PRODUCTS (product_id);
SQL>
SQL> create table CUSTOMERS
cont>     (customer_id  CUSTOMER_ID identity (1,1) primary key,
cont>      customer_name        char (100)
cont>     );
SQL> create unique index CUSTOMERS_IX on CUSTOMERS (customer_id);
SQL>
SQL> create table ORDERS
cont>     (order_id             ORDER_ID identity (1000) primary key,
cont>      order_date           timestamp,
cont>      customer_id  CUSTOMER_ID references CUSTOMERS
cont>      );
SQL> create unique index ORDERS_IX on ORDERS (order_id);
SQL>
SQL> create table ORDER_LINES
cont>     (order_id             ORDER_ID references ORDERS,
cont>      line_number          LINE_NUMBER,
cont>      product_id           PRODUCT_ID references PRODUCTS,
cont>      quantity             integer,
cont>      discount             float
cont>     );
SQL> create unique index ORDER_LINES_IX on ORDER_LINES (order_id, line_number);
SQL>
```

```
SQL> show sequences
Sequences in database with filename SQL$DATABASE
     CUSTOMERS
     ORDERS
     ORDER_LINES
     PRODUCTS
SQL> show sequences ORDERS
     ORDERS
 Sequence Id: 4
 Initial Value: 1000
 Minimum Value: 1000
 Maximum Value: (none)
 Next Sequence Value: 1000
 Increment by: 1
 Cache Size: 20
 No Order
 No Cycle
 No Randomize
 Wait
 Comment:       column IDENTITY sequence
```

As can be seen in the example, the START WITH value was explicitly set to 1000, but the INCREMENT BY value was defaulted to 1.

Example 2: Defaulting all attributes of IDENTITY sequence

```
SQL> create table PRODUCTS
cont>     (product_id           PRODUCT_ID identity primary key,
cont>      ...);
SQL> show sequence PRODUCTS
     PRODUCTS
 Sequence Id: 5
 Initial Value: 1
 Minimum Value: 1
 Maximum Value: (none)
 Next Sequence Value: 1
 Increment by: 1
 Cache Size: 20
 No Order
 No Cycle
 No Randomize
 Wait
 Comment:       column IDENTITY sequence
```

As can be seen in the example, both the START WITH and INCREMENT BY values for the sequence have defaulted to 1.

Example 3: Show that the IDENTITY sequence is reserved and cannot be dropped

```
SQL> drop sequence ORDERS;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-E-NOMETSYSREL, operation illegal on system defined metadata
-RDMS-E-SEQNOTDEL, sequence "ORDERS" has not been deleted
```

Example 4: Adding an identity column to an existing table

```
SQL> alter table EMPLOYEES
cont>   add column SEQUENCE_ID integer identity (1000, 10)
```

```
cont>   comment is 'Add unique sequence number for every employee';
SQL>
SQL> show table (column) EMPLOYEES
Information for table EMPLOYEES

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

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

5.1.10 Enhanced Bitmapped Scans

The bitmapped scan optimization used in conjunction with ranked indexes has been enhanced to handle more types of query selections.

Prior to this release, bitmapped scans were only performed if there were at least two ranked indexes that could be used to satisfy the query and if the query selection on each of the indexes were exact equalities.

This limited the usefulness of the bitmapped scan optimization to relatively simple queries.

Bitmapped scans have now been enhanced to allow the selection criteria to contain ranges: for example, queries using "OR", "IN", "LIKE", ">", etc.

For example, bitmapped scan may now be carried out on queries such as:

```
SQL> create index rsex on employees(sex) type sorted ranked;
SQL> create index rlast_name on employees(last_name) type sorted
ranked;

SQL> select last_name, sex from employees where (last_name  ='Toliver' or
cont> last_name =  'Smith') and (sex = 'M' or sex = 'F');

Leaf#01 FFirst EMPLOYEES Card=100       Bitmapped scan
  BgrNdx1 RSEX [(1:1)2] Fan=19
  BgrNdx2 RLAST_NAME [(1:1)2] Fan=12
 LAST_NAME        SEX
 Toliver          M
 Smith            M
 Smith            M
3 rows selected

SQL> select last_name, sex from employees where (last_name like
'Tol%') and
cont> (sex = 'M' or sex = 'F');
Leaf#01 FFirst EMPLOYEES Card=100       Bitmapped scan
  BgrNdx1 RLAST_NAME [1:1] Bool Fan=12
  BgrNdx2 RSEX [(1:1)2] Fan=19
 LAST_NAME        SEX
 Toliver          M
1 row selected
```

Bitmapped scans can also be carried out using indexes other than sorted ranked indexes as long as at least one index selected by the dynamic optimizer to access information from the subject table is a sorted ranked index.

This feature is available in Oracle Rdb Release 7.1.0.2.

5.1.11 Extended Record Compression

In previous versions, Oracle Rdb performed record compression by compressing runs of 3 or more repeating characters. A record to be compressed would be divided into sequences of repeating and non−repeating bytes.

The compression information itself is stored in a single byte. The high bit is set if the byte is indicating compression and clear if the byte is indicating no compression. The lower 7 bits contain the count. The count starts at 0; that is, if there is 1 byte in the run, the count will be 0.

If compression is taking place, the following byte will contain the repeating character. If compression is not taking place, the non−repeating run of bytes will be stored along with the count of non−repeating bytes.

For example:

```
            0022  03D4  line 1: record type 34
         00 0001  03D6  Control information
                  ....  8 bytes of static data
FE00008205000102  03D9   data '.......'
              00  03E1  padding '.'
```

This storage segment contains a single column of data type INTEGER (longword) with a value of 5. Contained in the storage segment (the 8 bytes of static data) is the following:

```
FE  00  00  82  05  0001  02
```

Reading from right to left (from the beginning of this record):

- 02 − Record compression. Bits 0−6 are the run length, in this case 3; bit 7 is clear so what follows is 3 non−repeating bytes.
- 0001 − The first two non−repeating bytes. The record version number, which is a word.
- 05 − Non−repeating byte. This is the first byte of the longword integer value.
- 82 − Record compression. Bits 0−6 are the run length, in this case 3; bit 7 is set so there is compression of the following 1 byte, which, when expanded, will be 3 bytes of 00 (null). These are the three high order bytes of the longword integer.
- 00 − Repeating byte, null value.
- 00 − Record compression. Bits 0−6 are the run length, and bit 7 is not set, therefore no compression. The count is zero, so the run length is 1 byte.
- FE − Null bit vector. 1 byte for each 8 columns in the table.

Because 7 bits are used to encode the length, and the maximum value stored in 7 bits is 127, the longest repeating or non−repeating run that can be encoded with one zero based compression byte is 128 bytes of data.

For very long runs of repeating bytes, this meant that Oracle Rdb had to use many compression bytes to encode the long string.

In Oracle Rdb Release 7.1.0, extended compression was introduced but the Release Note documenting it was inadvertently left out.

If the count is zero in a compression byte, this indicates extended compression, and the actual count is stored in a word following the compressed character.

The following small record is a single compressed varchar(3000) field containing the data value 'AAAA'. Note how the extended compression is used. The count 0BB3 is the count (2995+1) of repeating NULL bytes in the extended varchar field.

```
                            0028  03DC  line 0 (1:1971:0) record type 40
                         00 0001  03DE  Control information
                                  ....  13 bytes of static data
      FE000BB3008041830004000103  03E1   data '......A..³.. '

FE 00 0BB3 00 80 41 83 00040001 03
```

Reading from right to left (from the beginning of this record):

- 03 − Compression byte. The high order bit is set, indicating a run of non−repeating bytes, and the count is 3 indicating what follows is 4 non−repeating bytes.
- 00040001 − The four non−repeating bytes. In this case, the first two (0001) are the record version number, and the second two (0004) are the word length of the actual data in the varchar column.
- 83 − Compression byte. The high order bit is set, and the count is 3. This indicates a repeating run of four characters in length.
- 41 − The byte that repeats 4 times. In this case hex 41 is the ASCII character 'A'.
- 80 − Compression byte. The high order bit is set indicating a repeating run, but the count is zero indicating extended compression.

- 00 − The repeating byte (in this case null because a varchar with an actual length less than the defined length is padded with null characters out to the defined length).
- 0BB3 − The extended compression run length. Hex 0BB3 is the count (2995+1) of repeating NULL bytes in the extended varchar field.
- 00 − Compression byte. The high order bit is clear indicating a non−repeating run, and the count is zero indicating a run length of 1 byte.
- FE − Null bit vector.

In this way, far fewer compression bytes are needed for very long repeating strings.

Extended compression is enabled once a database has been converted to Oracle Rdb Release 7.1.0. If a database has been converted with the */NOCOMMIT* qualifier, extended compression will not be used until the convert is committed.

5.1.12 RMU /UNLOAD /AFTER_JOURNAL Wildcard Table Names

The RMU /UNLOAD /AFTER_JOURNAL command now supports wildcard processing of table names. The asterisk (*) and the percent sign (%) wildcard characters can be used in the table name specification to select all tables that satisfy the components you specify. The asterisk (*) matches zero or more characters and the percent sign (%) matches a single character.

Further, for table names that contain wildcard characters, if the first character of the string is a pound sign (#), the wildcard specification is changed to a "not matching" comparison. This allows exclusion of tables based on a wildcard specification. Note that the pound sign (#) designation is only evaluated when the table name specification contains an asterisk (*) or percent sign (%).

For example, a table name specification of "*" indicates that all tables in the database are to be selected. A table name specification of "FOO%" indicates that all table names that are four characters long and that start with the string "FOO" (such as "FOOD" and "FOOT") are to be selected.

This feature is available in Oracle Rdb Release 7.1.0.2.

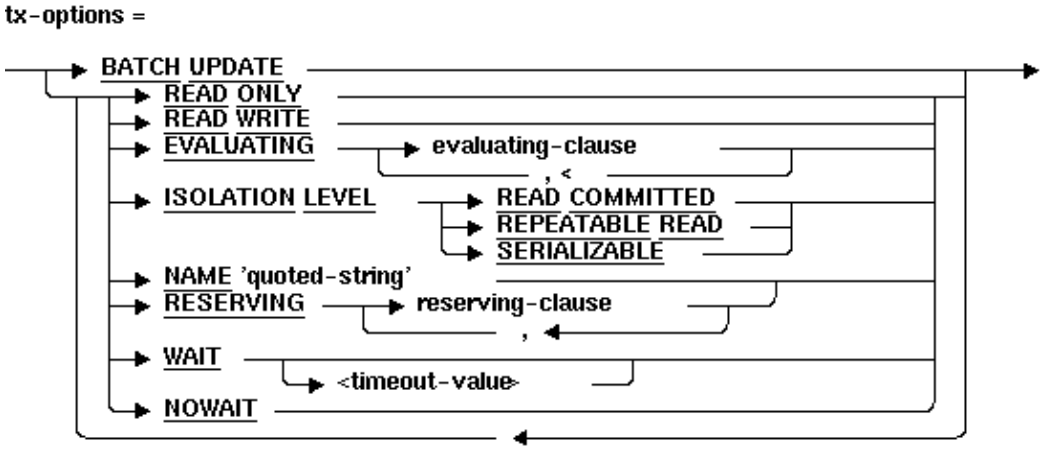
5.1.13 New NAME Clause for SET/DECLARE TRANSACTION Statement

This release of Oracle Rdb supports the NAME clause as part of the DECLARE and SET TRANSACTION statements so that the transaction can be given a title. This information is displayed by the SET FLAGS TRANSACTION keyword.

```
SQL> declare transaction read write name 'default−transaction';
SQL> select * from rdb$database;
~T Compile transaction (3) on db: 1
~T Transaction Parameter Block: (len=23)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_BUFFER_NAME "default−transaction"
0016 (00022) TPB$K_WRITE (read write)
~T Start_transaction (3) on db: 1, db count=1
.
.
.
```

This is the revised syntax for the tx−options clause for both SET TRANSACTION and DECLARE TRANSACTION.

FORMAT



Usage Notes

- Only one of the clauses: READ ONLY, READ WRITE or BATCH UPDATE may be used.
- No other clauses may be specified with BATCH UPDATE.
- Only one of the clauses, WAIT and NOWAIT, may be used.
- ISOLATION LEVEL may only be specified once.
- The clauses can be specified in any order.
- The quoted−string provided for NAME can be up to 255 octets in length.

5.1.14 New Built In Functions for Oracle RDBMS Compatibility

In prior releases of Oracle Rdb 7.1, the functions LENGTH and LENGTHB were provided as SQL stored functions that accepted a VARCHAR (2000) parameter and performed the appropriate CHARACTER_LENGTH or OCTET_LENGTH operation on the argument.

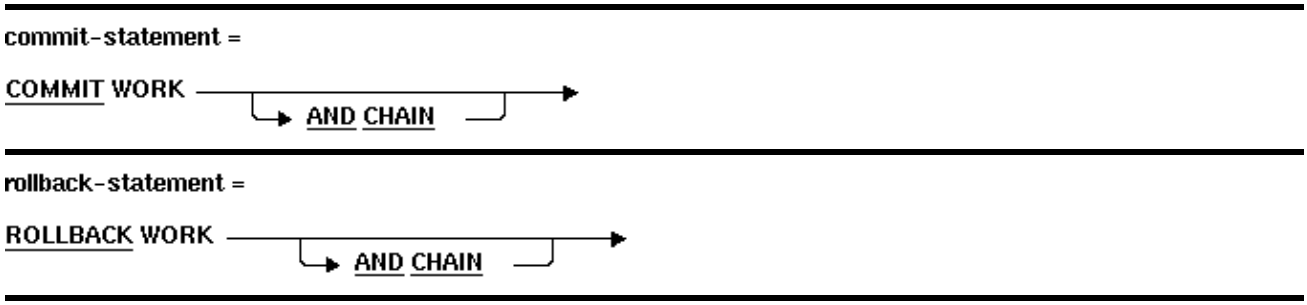
These functions, as supplied by SYS$LIBRARY:SQL_FUNCTIONS, are now obsolete. They are retained in the database for existing applications but new applications will now use new native functions in Rdb. These changes allow a more general usage for multiple characters sets and string length.

- LENGTH is a synonym for the ISO SQL:1999 function CHAR_LENGTH (or CHARACTER_LENGTH).
- LENGTHB is a synonym for the ISO SQL:1999 function OCTET_LENGTH.
- VSIZE is a synonym for the Rdb function SIZEOF.

Please refer to the Oracle Rdb7 SQL Reference Manual for a description of LENGTH, and LENGTHB. The Oracle Rdb Release 7.1.0 Release Notes describe the SIZEOF function.

5.1.15 New AND CHAIN Syntax Supported for COMMIT and ROLLBACK

Format



Usage Notes

- The AND CHAIN clause is only permitted in a compound statement (i.e. in a BEGIN ... END block), or as the body of a stored procedure.
- When AND CHAIN is used, a new transaction is implicitly started using the same attributes as the previously commited or rolled back transaction. Attributes such as READ WRITE, READ ONLY, RESERVING, EVALUATING, WAIT, and ISOLATION LEVEL are retained for the new transaction.
- Applications can use this new clause to simplify applications, since the complex transaction attributes need only be specified once.
- When the SET FLAGS option TRANSACTION_PARAMETERS is specified, a line of output is written to identify the chained transaction. Each SET TRANSACTION assigns a unique sequence number which is displayed after each transaction action line.

```
~T Restart_transaction (3) on db: 1, db count=1
```

Example

The following simple example executes SET TRANSACTION once at the start of the procedure. Then, periodically, the transaction is committed and restarted using the COMMIT AND CHAIN syntax. This simplifies the application since there is only one definition of the transaction characteristics.

```
SQL> -- process table in batches
SQL>
SQL> set compound transactions 'internal';
SQL> set flags 'transaction,trace';
SQL>
SQL> begin
cont> declare :counter integer = 0;
cont> declare :emp integer;
cont>
cont> set transaction
cont>     read write
cont>     reserving employees for exclusive write;
cont>
cont> for :emp in 0 to 600
cont> do
cont>     begin
cont>     declare :id char(5)
cont>           default substring (cast (:emp+100000 as varchar(6))
```

```
cont>                               from 2 for 5);
cont>     if exists (select * from employees where employee_id = :id)
cont>     then
cont>         trace 'found: ', :id;
cont>         if :counter > 20
cont>         then
cont>             commit and chain;
cont>             set :counter = 1;
cont>         else
cont>             set :counter = :counter + 1;
cont>         end if;
cont>     end if;
cont>     end;
cont> end for;
cont>
cont> commit;
cont> end;
~T Compile transaction (1) on db: 1
~T Transaction Parameter Block: (len=2)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_WRITE (read write)
~T Start_transaction (1) on db: 1, db count=1
~T Rollback_transaction on db: 1
~T Compile transaction (3) on db: 1
~T Transaction Parameter Block: (len=14)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_WRITE (read write)
0002 (00002) TPB$K_LOCK_WRITE (reserving) "EMPLOYEES" TPB$K_EXCLUSIVE
~T Start_transaction (3) on db: 1, db count=1
~Xt: found: 00164
~Xt: found: 00165
.
.
.
~Xt: found: 00185
~T Commit_transaction on db: 1
~T Prepare_transaction on db: 1
~T Restart_transaction (3) on db: 1, db count=1
~Xt: found: 00186
~Xt: found: 00187
.
.
.
~Xt: found: 00435
~Xt: found: 00471
~T Commit_transaction on db: 1
~T Prepare_transaction on db: 1
SQL>
```

5.1.16 New Options for SET FLAGS Statement

This release of Oracle Rdb adds two new SET FLAGS keywords.

- WATCH_CALL
  This keyword traces the execution of queries, triggers and stored functions and procedures. The output includes the name of the trigger, function or procedure or "unnamed" for an anonymous query. In most cases, a query can be named using the OPTIMIZE AS clause. It also includes the value of

CURRENT_USER during the execution of that routine. CURRENT_USER may be inherited from any module that uses the AUTHORIZATION clause.
This flag can be disabled using NOWATCH_CALL, by using SET NOFLAGS, or using SET FLAGS 'NONE'.

- WATCH_OPEN
  This keyword traces all queries executed on the database. This may include SQL runtime system queries to lookup table names, etc as well as queries executed by the application. The output includes the 32 digit hex identifier, the same as used by the CREATE OUTLINE statement. This value uniquely identifies the query being executed.
  If a query is a stored routine (function or procedure) then the notation "(stored)" is appended. If the query is named then it will be classified as "(query)", otherwise it will be designated as "(unnamed)". See the examples below for details.
  This flag can be disabled using NOWATCH_OPEN, by using SET NOFLAGS, or using SET FLAGS 'NONE'.

Usage Notes

- These keywords can also be used with the RDMS$SET_FLAGS logical name.
- The RDMS$DEBUG_FLAGS value "Xa" can also be used to enable WATCH_CALL.
- The RDMS$DEBUG_FLAGS value "Xo" can also be used to enable WATCH_OPEN.
- When using interactive or dynamic SQL, both WATCH_CALL and WATCH_OPEN will generate trace lines for the queries performed by the SQL runtime system against the Rdb system tables. There is no mechanism to disable the trace of such information.
- These flags cause queries and routines to be modified to output this information. This might add some extra CPU overhead to the application while this information is collected. Even when the flags are disabled, there exists some overhead that is not eliminated until the module or query is released, usually at DISCONNECT time.

Examples

Example 1: WATCH_CALL

This example shows the output of WATCH_CALL for an INSERT statement which causes an AFTER INSERT trigger (AFTER_INSERT) to be executed which calls a SQL function WRITE_TEXT to trace the input data. It then traces a query named using the OPTIMIZE AS clause.

```
SQL> insert into SAMPLE_T values ('Fred');
~Xa: routine "(unnamed)", user=SMITHI
~Xa: routine "AFTER_INSERT", user=SMITHI
~Xa: routine "WRITE_TEXT", user=SMITHI
~Xt: Fred
1 row inserted
SQL> select * from SAMPLE_T
cont>     optimize as LOOKUP_SAMPLE_T;
~Xa: routine "LOOKUP_SAMPLE_T", user=SMITHI
 NEW_NAME
 Fred
1 row selected
```

Example 2: WATCH_OPEN

This example shows the output of WATCH_OPEN for the same INSERT statement as seen in Example 1.

```
SQL> insert into SAMPLE_T values ('Fred');
~Xo: Start Request B667E51E3625026EB7FFF3F4D3A16DC3 (unnamed)
~Xo: Start Request A8568053FE5A1A0852A1BE83A884016F "AFTER_INSERT" (query)
~Xo: Start Request 08AE59062657299B4768F6C2DFB6928E "WRITE_TEXT" (stored)
~Xt: Fred
1 row inserted
SQL>
SQL> select * from SAMPLE_T
cont>     optimize as LOOKUP_SAMPLE_T;
~Xo: Start Request F6025FAB1DD36B0DE0E52F3A9641BC5F "LOOKUP_SAMPLE_T" (query)
 NEW_NAME
 Fred
 Fred
2 rows selected
```

5.2 Enhancements Provided in Oracle Rdb Release 7.1.0.1

5.2.1 SQL Now Supports a Native ABS Function

In prior releases of Oracle Rdb, the ABS function was provided by the SQL_FUNCTIONS script. This function was a DOUBLE PRECISION function that allowed values of most data types to be processed.

However, there were some inconsistencies introduced when large BIGINT values were used as rounding errors were introduced since DOUBLE PRECISION supports about 16 digits accuracy compared to the 18 digits supported by BIGINT. In addition, the INTERVAL data type could not be used with the provided function.

With this release, a new conditional function, ABS, conforming to the SQL:1999 database language standard, is now available. The ABS function returns NULL if the passed value expression evaluates to NULL. The datatype of the result is the same as the passed value expression and supports scaled values of these data types: TINYINT, SMALLINT, INTEGER, BIGINT, REAL, FLOAT, DOUBLE PRECISION, INTERVAL, DECIMAL, NUMERIC and NUMBER.

The absolute value function (ABS) returns NULL if the value expression evaluates to NULL. If the value expression evaluates to a value less than zero then that value is negated so that a positive value is returned. Otherwise the value is returned unchanged. For instance, ABS (−1) will return the value 1.

ABS (a) is equivalent to the CASE expression:

```
case
    when a < 0 then − a
    else a
end
```

USAGE NOTES:

- The SQL_FUNCTIONS script still includes the ABS external function definition for those stored definitions (procedures, functions, triggers, views, etc.) or compiled applications that currently use it. However, new references to ABS will use the new builtin conditional expression.
- Applications wishing to continue to use the external function should use delimiters around the ABS function name, as in the following example.

  ```
  SQL> set quoting rules 'SQL92';
  SQL> select "ABS" (v) from T;
  ```

  The delimited name will force the function definition to be used.
- Please refer to Appendix G, Oracle Rdb7 SQL Reference Manual, Volume 3 for more information on the SQL_FUNCTIONS script.

Example 1: This example uses the ABS function on an INTERVAL result of a date subtraction.

```
SQL> select
```

```
cont>    ABS ((birthday − current_date) year(3))
cont> from employees
cont> order by employee_id
cont> limit to 10 rows;

  054
  047
  047
  064
  068
  062
  044
  069
  050
  074
10 rows selected
```

Example 2: This shows a more complex use of ABS within a statistical function.

```
SQL> −− what is the average time in a job for each employee
SQL> −− exclude anyone on there first job
SQL> select
cont>    employee_id,
cont>    AVG (ABS (EXTRACT (MONTH FROM (job_start − job_end) month (4))))
cont>           as "Average Job" edit using '−−,−−−,−−9.99" years"'
cont> from JOB_HISTORY
cont> where employee_id < '00200'
cont> group by employee_id
cont> having COUNT (*) > 1;
 EMPLOYEE_ID           Average Job
 00164                 14.00 years
 00165                 22.67 years
 00166                 20.00 years
 00167                 14.50 years
 00168                 26.33 years
 00169                 22.67 years
...etc...
 00197                 26.33 years
 00198                 37.00 years
 00199                 35.00 years
30 rows selected
%RDB−I−ELIM_NULL, null value eliminated in set function
```

5.2.2 New DUMP Output Format for LogMiner

A new output format type of "DUMP" has been added to the RMU /UNLOAD /AFTER_JOURNAL command. This output format is intended solely as a debug and informational tool. For each column of a record, the first 200 bytes of data contents are formatted such that binary numeric fields are converted to text and text fields are displayed with periods (.) replacing non−printable characters. NULL columns are indicated with the character string "NULL". The actual data length is indicated for VARCHAR columns.

Example output with the /FORMAT=DUMP qualifier:

```
$ RMU /UNLOAD /AFTER_JOURNAL MYDB.RDB MYDB.AIJBCK /FORMAT=DUMP
    /TABLE=(NAME=ALL_DATATYPES_TBL, OUTPUT=SYS$OUTPUT:)
RDB$LM_ACTION                   : M
RDB$LM_RELATION_NAME            : ALL_DATATYPES_TBL
```

```
RDB$LM_RECORD_TYPE              : 25
RDB$LM_DATA_LEN                 : 460
RDB$LM_NBV_LEN                  : 66
RDB$LM_DBK                      : 46:635:0
RDB$LM_START_TAD                : 21-JUL-2001 15:48:52.6512009
RDB$LM_COMMIT_TAD               : 21-JUL-2001 15:48:53.0586846
RDB$LM_TSN                      : 160
RDB$LM_REC_VER                  : 1
TINT                            : -123
SINT                            : -321
INTEGER                         : -212
BINT                            : NULL
DECIMAL                         : -145
NUMERIC                         : NULL
FLOAT                           : -1.000000000000000E+000
DOUBLE_PRECISION                : -2.000000000000000E+000
CHAR1                           : A
CHAR20                          : ABCDEFGHIJKLMNOPQRST
VCHAR_COL                       : (10) ABCDEFGHIJ
```

Note

The contents and format of the output when the /FORMAT=DUMP qualifier is specified may change in the future.

If needed, the record definition (.RRD) file may be used to determine the actual data type for each field of the table(s) being extracted.

5.2.3 Data and SPAM Prefetch Screens Added to RMU/SHOW STATISTICS

Two new screens have been added to the PIO statistics part of RMU/SHOW STATISTICS. These screens display prefetch statistics (APF and DAPF). In prior versions, the DAPF statistics were displayed on the "Fetch" screens. Those statistics were moved to the new prefetch screens. In addition, APF statistics are now displayed on the new screens as well. An example is provided below:

```
Node: NODE1 (1/1/1)     Oracle Rdb V7.0-62 Perf. Monitor  6-AUG-2001 10:28:10.65
Rate: 3.00 Seconds      PIO Statistics--Data Prefetches     Elapsed: 00:58:17.86
Page: 1 of 1                    DEV:[DIR]DB.RDB                     Mode: Online
------------------------------------------------------------------------------

statistic.........      rate.per.second.............. total....... average......
name..............      max..... cur..... avg....... count....... per.trans....

APF start:success              0       0       0.4          872           1.0
         :failure              0       0       0.0          101           0.1

APF I/O: utilized              0       0       0.4          872           1.0
       : wasted                0       0       0.0            0           0.0


DAPF start:success             0       0       0.0           74           0.0
          :failure             0       0       0.0           62           0.0

DAPF I/O: utilized             0       0       0.0           18           0.0
        : wasted               0       0       0.0           56           0.0
```

The information on these screens may be used to determine the effectiveness of the APF and DAPF features. The individual rows may be interpreted as follows:

- The "APF start:success" statistics shows how many times Oracle Rdb successfully initiated an I/O to prefetch a buffer.
- The "APF start:failure" statistics shows how many times Oracle Rdb attempted to initiate a prefetch but was unable to obtain the necessary buffer lock to proceed.
- The "APF I/O: utilized" statistics shows how many times Oracle Rdb actually used a buffer that was prefetched.
- The "APF I/O: wasted" statistics shows how many times Oracle Rdb prefetched a buffer but never actually used it.

5.2.4 RMU/SHOW STATISTICS Stall Log Lock Information Optional

Bug 1704232

A new optional keyword "[NO]LOG_STALL_LOCK" has been added to the "/OPTIONS" qualifier of the RMU/SHOW STATISTICS command. When using the /STALL_LOG qualifier to write stall messages to a log file, you can now specify /OPTIONS=NOLOG_STALL_LOCK to prevent lock information from being written to the log file.

The following example shows stall log information first with the lock information and then without the lock information:

```
$ RMU /SHOW STATISTICS /NOINTERACTIVE /STALL_LOG=SYS$OUTPUT: -
    DUA0:[DB]MFP.RDB
 Oracle Rdb X7.1-00 Performance Monitor Stall Log
 Database DPA500:[RDB_RANDOM.RDB_RANDOM_TST_247]RNDDB.RDB;1
 Stall Log created  4-SEP-2001 11:27:03.96
11:27:03.96 0002B8A1:1 11:27:03.67 waiting for record 118:2:2 (PR)
   State... Process.ID Process.name... Lock.ID. Rq Gr Queue "record 118:2:2"
   Blocker: 000220A7   RND_TST_24716   0F019E52    EX Grant
   Waiting: 0002B8A1   RND_TST_24715   4500C313 PR    Wait
11:27:03.96 0002B8A8:1 11:27:02.32 waiting for record 101:3:0 (EX)
   State... Process.ID Process.name... Lock.ID. Rq Gr Queue "record 101:3:0"
   Blocker: 000220AD   RND_TST_24710   0B00176A    PR Grant
   Blocker: 000220A7   RND_TST_24716   52018A3F    PR Grant
   Waiting: 0002B8A8   RND_TST_2474    3C00B5AF EX PR Cnvrt
11:27:03.96 0002B89C:1 11:27:00.15 waiting for record 114:4:1 (PR)
   State... Process.ID Process.name... Lock.ID. Rq Gr Queue "record 114:4:1"
   Blocker: 000220A7   RND_TST_24716   180033CC    EX Grant
   Waiting: 0002B89C   RND_TST_2479    110066BA PR    Wait

$ RMU /SHOW STATISTICS /NOINTERACTIVE /STALL_LOG=SYS$OUTPUT: -
    DUA0:[DB]MFP.RDB /OPTIONS=NOLOG_STALL_LOCK
 Oracle Rdb X7.1-00 Performance Monitor Stall Log
 Database DPA500:[RDB_RANDOM.RDB_RANDOM_TST_247]RNDDB.RDB;1
 Stall Log created  4-SEP-2001 11:28:34.68
11:28:34.69 0002B8B8:1 11:28:33.69 waiting for logical area 146 (PR)
11:28:34.69 0002B8A8:1 11:28:32.76 waiting for record 114:4:2 (PR)
11:28:34.69 0002B8B3:1 11:28:33.06 waiting for record 114:4:2 (PR)
11:28:34.69 0002B8B0:1 11:28:31.96 waiting for record 111:7:7 (EX)
```

5.2.5 New Option for the GET DIAGNOSTICS Statement

For Oracle Rdb Release 7.1.0.1, a new option has been added to the GET DIAGNOSTICS statement: IMAGE_NAME.

This keyword requests that the activating image name be returned to the caller. The image name includes the node name from which the attach was started. This might be a node different than that on which the Rdb server is running.

The data is returned to the caller as a VARCHAR (255) value and should be assigned to either a VARCHAR or CHAR data type that supports the ASCII character set.

The following example uses a SQL procedure to fetch the image name for the currently running application (in this case interactive SQL).

```
SQL> set flags 'trace';
SQL> begin
cont> declare :i varchar(512);
cont> get diagnostics :i = image_name;
cont> trace char_length (:i);
cont> trace '"' || :i || '"';
cont> end;
~Xt: 57
~Xt: "MYNODE::$111$DUA618:[SYS0.SYSCOMMON.][SYSEXE]SQL$71.EXE;1"
```

5.2.6 Alternate Outline Ids

If outlines have not been disabled, Oracle Rdb will search for an appropriate outline for the query it is optimizing, thus allowing some user control of the strategy used for execution of a query.

The OPTIMIZE USING clause may be used to tell the optimizer which outline to use for compilation. If no OPTIMIZE USING clause is present, Rdb uses the query to generate an identifier which it will use to try to locate an appropriate outline.

In many situations, such as when using third party software, it is not possible for the user to provide an outline name for the query and thus the only alternative Rdb had was to try to locate an outline with a matching identifier.

As the identifier is a hashed value that depends on the query structure, small changes in the query, such as different literal values, can change the identifier produced as in the following example.

```
SQL> set flags 'outline';
SQL> select * from employees where employee_id = '1';
-- Rdb Generated Outline : 19-SEP-2001 13:52
create outline QO_8797A75D6D03F6BD_00000000
id '8797A75D6D03F6BDD211A092CE6F3A2C'
mode 0
as (
  query (
-- For loop
    subquery (
      EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
```

```
      )
    )
  )
compliance optional     ;
0 rows selected
SQL> select * from employees where employee_id = '9999';
-- Rdb Generated Outline : 19-SEP-2001 13:52
create outline QO_C9F12D27AC5D3163_00000000
id 'C9F12D27AC5D3163907A4329FDC8170A'
mode 0
as (
  query (
-- For loop
    subquery (
      EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
      )
    )
  )
compliance optional     ;
```

In this example, the two queries are optimized the same but the differing outline identifiers means that two different outlines would have to be created to control each query.

Oracle Rdb has now been enhanced to allow the optional creation of alternate outline identifiers. In this release, the optimizer discards literal values when producing the identifiers.

A new SET FLAGS attribute has been introduced to allow the control of these alternate identifiers, using either the SQL SET FLAGS statement or the RDMS$SET_FLAGS logical name.

```
ALTERNATE_OUTLINE_ID(LITERALS)
```

This attribute is not case sensitive and may be abbreviated to:

```
ALT(LIT)
```

The following example uses SET FLAGS to enable alternate query identifiers:

```
SQL> set flags 'alt(LIT), outline';
SQL> select * from employees where employee_id = '1';
-- Rdb Generated Outline : 19-SEP-2001 13:52
create outline QO_847AD7287E247D37_00000000
id '847AD7287E247D37E8E4CC8221FFC12E'
mode 0
as (
  query (
-- For loop
    subquery (
      EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
      )
    )
  )
compliance optional     ;
0 rows selected
SQL> select * from employees where employee_id = '9999';
-- Rdb Generated Outline : 19-SEP-2001 13:52
create outline QO_847AD7287E247D37_00000000
id '847AD7287E247D37E8E4CC8221FFC12E'
mode 0
```

```
as (
  query (
-- For loop
    subquery (
      EMPLOYEES 0     access path index       EMP_EMPLOYEE_ID
      )
    )
  )
compliance optional     ;
0 rows selected
```

Note that now the two outlines have the same identifier and the user may now store this more generic outline to be used by any similar query where only the literal values differ. For example:

```
SQL> set flags 'alt(lit)';
SQL> create outline o1 from (select * from employees where employee_id = '1');
SQL> set flags 'strat';
SQL> select * from employees where employee_id = '1';
~S: Outline "O1" used
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
SQL> select * from employees where employee_id = 'AAAAAA';
~S: Outline "O1" used
Conjunct        Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
```

Any outline stored for a query without the ALTERNATE_OUTLINE_ID flag being set will be created using the full query as in previous versions and will take precedence over any generic outline. For example:

```
SQL> set noflags;
SQL> create outline o1 from (select * from employees where employee_id = '1');
SQL> set flags 'strat';
SQL> select * from employees where employee_id = '1';
~S: Outline "O1" used
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
SQL> select * from employees where employee_id = '9999';
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
SQL> set noflags;
SQL> set flags 'alternate(lit),nooutline';
SQL> create outline o2 from (select * from employees where employee_id = '1');
SQL>
SQL> set flags 'strat';
SQL> select * from employees where employee_id = '1';
~S: Outline "O1" used
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
SQL> select * from employees where employee_id = '9999';
~S: Outline "O2" used
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
SQL>
SQL> set flags 'noalt';
```

```
SQL> select * from employees where employee_id = '1';
~S: Outline "O1" used
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
SQL> select * from employees where employee_id = '9999';
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
SQL> drop outline o1;
SQL> set flags 'alt(literals)';
SQL> select * from employees where employee_id = '1';
~S: Outline "O2" used
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
SQL> select * from employees where employee_id = '9999';
~S: Outline "O2" used
Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [1:1]      Direct lookup
0 rows selected
```

As shown in the previous example, Oracle Rdb will try to locate an outline using the more generic identifier only if the ALTERNATE_OUTLINE_ID flag has been set.

The ALTERNATE_OUTLINE_ID flag is not set by default and must be explicitly set using either SET FLAGS or the RDMS$SET_FLAGS logical.

This feature is available in Oracle Rdb Release 7.1.0.1.

5.2.7 Field Widths Wider on Row Cache Overview Display

On the "Row Cache Overview" display, the width of the "Searches" column has been increased from 9 to 10 characters to allow a display of up to 4294967295 (after this value, the 32−bit counter wraps back to zero). In addition, the width of the cache name column is tied to the screen width. If the screen is set to be wide enough (over 90 columns), the full cache name will be displayed; normally, only the first 24 characters of the name are displayed.

Additionally, the comparison used when sorting by values on the "Row Cache Overview" display has been modified to be unsigned (rather than signed). This prevents some cases of very large values being sorted in an incorrect order.

5.2.8 FOR Counted Loop Enhancements

In Oracle Rdb Release 7.1, the FOR counted loop was added to SQL. This type of loop increments a declared variable from an initial value to a final value. In the prior release of Rdb, the data type of the variable had to be a numeric data type (TINYINT, SMALLINT, INTEGER, BIGINT, REAL, FLOAT, DOUBLE PRECISION, NUMBER, NUMERIC, or DECIMAL).

The following enhancements have been made for this release:

- The following data types are now also legal for this type of FOR loop.

  ```
  INTERVAL YEAR
  ```

```
INTERVAL MONTH
INTERVAL DAY
INTERVAL HOUR
INTERVAL MINUTE
INTERVAL SECOND
```

If INTERVAL is used, then the initial and final values must be of the same type (i.e. the expressions must have the same data type as the loop variable).

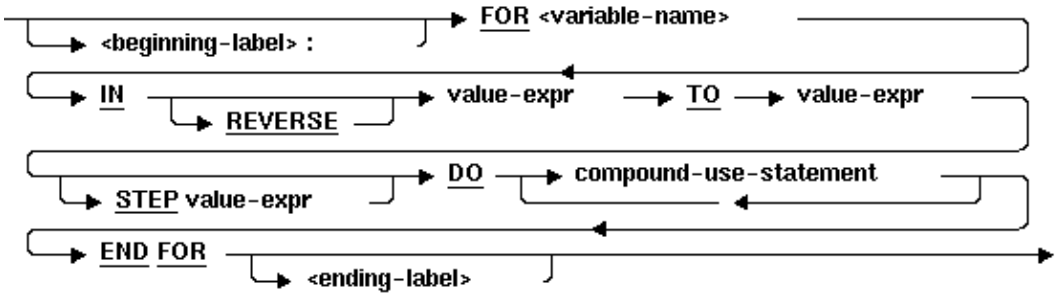
- The data type rules for the initial and final values have been relaxed when the loop variable is numeric. These value expressions can be any compatible numeric data type. For instance, floating point or scaled numeric values can now be used.
- A new optional STEP clause has been added to control the size of the increment between loop iterations. The step size is specified using a numeric value expression.

```
SQL> begin
cont> declare :i integer;
cont> for :i in 1 to 20 step 5
cont> do
cont>     trace :i;
cont> end for;
cont> end;
~Xt: 1
~Xt: 6
~Xt: 11
~Xt: 16
```

NOTE: Even if the loop control variable is an INTERVAL type, the STEP must be numeric type. In addition, the value must be greater than zero: use the REVERSE keyword to decrement the loop control variable.

FORMAT

counted-for-statement =



USAGE NOTES

- The FOR loop uses the keyword TO as a separator between the initial and final value expressions. This same keyword is used to separate the field names in an interval qualifier. Therefore, there is an ambiguity possible when an apparently well−formed expression is used.

```
SQL> begin
cont> declare :i interval year;
cont> for :i in interval'1' year to interval'4'year
```

```
for :i in interval'1' year to interval'4'year
                              ^
%SQL-W-LOOK_FOR_STT, Syntax error, looking for:
%SQL-W-LOOK_FOR_CON,            MONTH,
%SQL-F-LOOK_FOR_FIN,    found INTERVAL instead
```

This occurs because the TO separator is interpreted as part of the INTERVAL literal or expression. Programmers must enclose the initial expression in parentheses to avoid this ambiguity if it ends with an interval qualifier.

- The STEP value expression is evaluated before the loop variable is assigned a value. The value must be greater than zero and never NULL. If these constraints are violated, a runtime error is reported as shown in this simple example.

```
SQL> begin
cont> declare :l, :s integer;
cont>
cont> -- set the step size
cont> set :s = 0;
cont>
cont> for :l in reverse 1 to 10 step :s
cont> do
cont>     trace :l;
cont> end for;
cont> end;
%RDB-E-NOT_VALID, validation on field STEP caused operation to fail
SQL>
```

Example 1: This example shows an INTERVAL type as the loop variable.

```
SQL> begin
cont> declare :i interval year;
cont> for :i in (interval'1' year) to (interval'4'year)
cont> do
cont>     trace :i;
cont> end for;
cont> end;
~Xt:  01
~Xt:  02
~Xt:  03
~Xt:  04
```

Example 2: This example uses a complex expression as the STEP expression.

```
SQL> begin
cont> declare :i interval year;
cont> declare :k interval year = interval'18'year;
cont> declare :j integer = 2;
cont>
cont> for :i in (interval'1' year) to :k/2 step :j*2
cont> do
cont>     trace :i;
cont> end for;
cont> end;
~Xt:  01
~Xt:  05
```

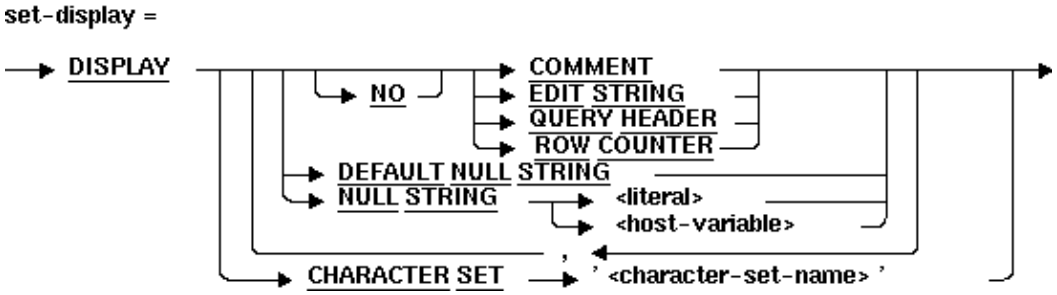
```
~Xt:  09
```

5.2.9 Enhancements to SET DISPLAY Statement for Interactive SQL

This release of Oracle Rdb, 7.1.0.1, includes the following enhancements to the SET DISPLAY statement.

- A new NULL STRING clause to change the way NULL values are displayed by interactive SQL.
- A new DEFAULT NULL STRING clause to revert to using the text 'NULL'.
- A new [NO] COMMENT clause to disable or enable the display of comment text by other SHOW commands (e.g. SHOW TABLE).

FORMAT



USAGE NOTES

- The width of the displayed column is calculated using the maximum of the length of the column name, the length of the QUERY HEADER, the length of the NULL string and the size of the formatted data.
- The statement SET DISPLAY DEFAULT NULL STRING is equivalent to SET DISPLAY NULL STRING 'NULL'.
- The SET NULL statement has been added for compatibility with Oracle SQL*Plus. SET NULL is a synonym for SET DISPLAY NULL STRING '', and SET NULL 'literal' is equivalent to SET DISPLAY NULL 'literal'.
- SET DISPLAY NULL STRING accepts a string literal, or a declared local variable.
- SHOW DISPLAY now displays the current NULL string.

```
SQL> show display
Output of the query header is enabled
Output of the row counter is enabled
Output using edit strings is enabled
Page length is set to 30 lines
Line length is set to 80 bytes
Display NULL values using "NULL"
```

- The GET ENVIRONMENT statement now includes the NULL_STRING keyword that can be used to save the currently defined text.

Example 1: Replace the NULL values with text to make the output easier to read.

```
SQL> select job_start, job_end,
cont>             (select department_name
```

```
cont>              from departments d
cont>              where d.department_code = jh.department_code)
cont> from job_history jh
cont> where employee_id = '00164';
 JOB_START     JOB_END
 21-Sep-1981   NULL          Board Manufacturing North
  5-Jul-1980   20-Sep-1981   Cabinet & Frame Manufacturing
2 rows selected
SQL> set display null string '(still employeed)'
SQL> select job_start, job_end,
cont>             (select department_name
cont>              from departments d
cont>              where d.department_code = jh.department_code)
cont> from job_history jh
cont> where employee_id = '00164';
 JOB_START           JOB_END
 21-Sep-1981         (still employeed)   Board Manufacturing North
  5-Jul-1980         20-Sep-1981         Cabinet & Frame Manufacturing
2 rows selected
```

Example 2: Disable the comment display to make the output of SHOW easier to read.

```
SQL> show domain id_dom
ID_DOM                          CHAR(5)
 Comment:       standard definition of employee id
SQL> set display no comment;
SQL> show domain id_dom
ID_DOM                          CHAR(5)
SQL>
```

Example 3: Save the current NULL string using GET ENVIRONMENT and restore after executing a query.

```
SQL> declare :ns varchar(100);
SQL> get environment (session) :ns = NULL_STRING;
SQL> set null;
SQL> select job_start, job_end,
cont>             (select department_name
cont>              from departments d
cont>              where d.department_code = jh.department_code)
cont> from job_history jh
cont> where employee_id = '00164';
 JOB_START     JOB_END
 21-Sep-1981                 Board Manufacturing North
  5-Jul-1980   20-Sep-1981   Cabinet & Frame Manufacturing
2 rows selected
SQL> set display null string :ns;
SQL> select job_start, job_end,
cont>             (select department_name
cont>              from departments d
cont>              where d.department_code = jh.department_code)
cont> from job_history jh
cont> where employee_id = '00164';
 JOB_START     JOB_END
 21-Sep-1981   NULL          Board Manufacturing North
  5-Jul-1980   20-Sep-1981   Cabinet & Frame Manufacturing
2 rows selected
```

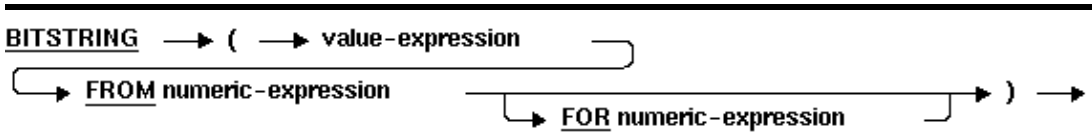
5.2.10 New BITSTRING Built In Function

Rdb now supports a BITSTRING function that can be used to extract selected bits from a binary data value. This functionality is primarily intended to query the bit values stored in the RDB$FLAGS columns in the Rdb system table but can also be used for user data.

BITSTRING accepts numeric and date/time values and processes them as bit arrays. The first (least significant) bit is numbered 1. The most significant bit depends on the data type.

- TINYINT has 8 bits
- SMALLINT has 16 bits
- INTEGER has 32 bits
- BIGINT, DATE, TIME, TIMESTAMP and INTERVAL types have 64 bits

FORMAT



USAGE NOTES

- The numeric expression after the FOR and FROM keywords must be an unscaled numeric value.
- If the numeric expression of the FOR clause is less than or equal to zero then it will be assumed equal to 1.
- If the FOR clause is omitted, it will default to a value that includes all remaining bits of the source value.
- If the FOR clause specifies a larger value than the number of bits remaining in the source then it will only return the remaining bits.

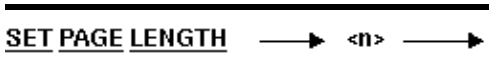
Example: Bit 1 in the RDB$FLAGS column of RDB$RELATIONS indicates that the table is a view. This example uses this query to fetch the names of all user defined views in the PERSONNEL database.

```
SQL> select rdb$relation_name
cont> from rdb$relations
cont> where rdb$system_flag = 0 and
cont>     bitstring (rdb$flags from 1 for 1) = 1;
 RDB$RELATION_NAME
 CURRENT_JOB
 CURRENT_SALARY
 CURRENT_INFO
3 rows selected
SQL>
```

5.2.11 New SET PAGE LENGTH Command for Interactive SQL

SQL now includes a SET PAGE LENGTH statement to size the page. Currently this is only used by the pagination control in the SQL HELP command.

FORMAT



USAGE NOTES

- The integer value must be a value between 10 and 32767.
- SET PAGE LENGTH can be used to effectively disable the paging performed by help by setting the length to a high value such as 32000.
- The page length is automatically set upon entry to interactive SQL and is based on the OpenVMS terminal setting for this session.
- The SHOW DISPLAY command can be used to view the currently defined page length.

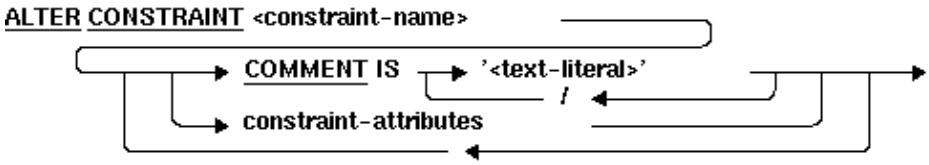
This example uses the SET PAGE LENGTH command to change the pagination length of HELP.

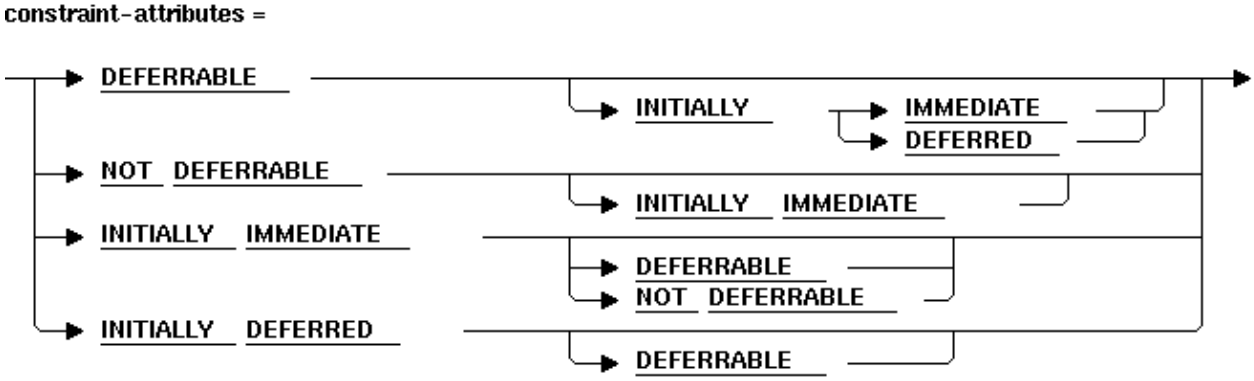
```
SQL> set page length 40;
SQL> show display
Output of the query header is enabled
Output of the row counter is enabled
Output using edit strings is enabled
Page length is set to 40 lines
Line length is set to 80 bytes
Display NULL values using "NULL"
```

5.2.12 New ALTER CONSTRAINT Statement

Oracle Rdb Release 7.1 includes an ALTER CONSTRAINT statement.

FORMAT





Note: constraint−attributes are described in the Oracle Rdb New and Changed Features Manual.

USAGE NOTES

- If a constraint attribute is changed, it will only be effective for future references to the table containing that constraint. That is, if a constraint is already active then it will use the previously defined attributes.
- The constraint name can be prefixed with an alias name as in the following example.

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

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

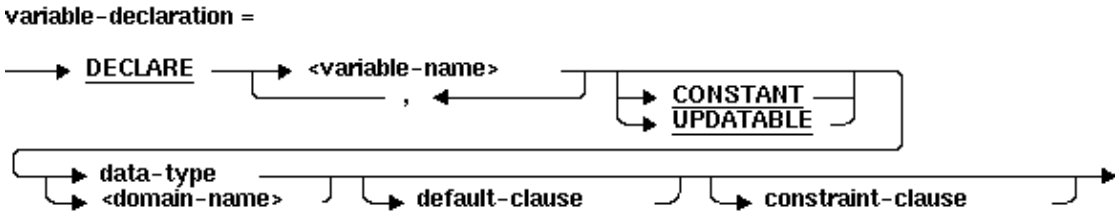
```
SQL> set dialect 'sql99';
SQL> attach 'file db$:mf_personnel';
SQL>
SQL> create table PERSON
cont>     (last_name  char(20)
cont>         constraint MUST_HAVE_LAST_NAME
cont>             not null
cont>             deferrable,
cont>      first_name char(20),
cont>      birthday   date
cont>      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


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

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

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

Constraints referencing table PERSON:
No constraints found
```

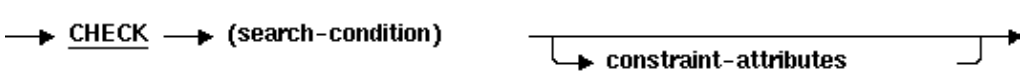
```
SQL>
SQL> alter constraint ALL_UNIQUE
cont>     deferrable initially deferred;
SQL>
SQL> alter constraint MUST_HAVE_LAST_NAME
cont>     comment is 'We must assume all persons have a name'
cont>     not deferrable;
SQL>
SQL> alter constraint MUST_BE_IN_PAST
cont>     deferrable initially immediate;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


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

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

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

Constraints referencing table PERSON:
No constraints found

SQL>
SQL> commit;
```

5.2.13 DECLARE Variable Now Supports CHECK Constraint

Variables declared within a compound statement (BEGIN...END) can now include a CHECK constraint to prevent out of range assignments to variables.

FORMAT





USAGE NOTES

- The constraint−clause is applied to all variables listed in DECLARE. The keyword VALUE can be used as a placeholder for the variable name with SQL correctly applying the constraint to all variables.
- Only the NOT DEFERRABLE and INITIALLY IMMEDIATE syntax is supported for variable constraints. This is also the default if no constraint−attributes are specified.
- A runtime error is signaled if the constraint is violated. This error will include the name of the variable.
- When a DEFAULT is not used in the declare statement, the contents of the variable are undefined. Therefore, any variable that uses a CHECK constraint must also provide a DEFAULT clause to ensure that the variable's value is consistent.
- Currently module global variables do not support constraints. This is planned for a future release of Oracle Rdb.

The following example shows the use of a CHECK constraint to prevent illegal values being assigned to control variables for a REPEAT loop. The singleton SELECT will actually return zero to the local variable P which will cause a variable validation to fail.

```
SQL> begin
cont> declare :v integer = 0 check (value is not null);
cont> declare :p integer = 1 check (value is not null and value <> 0);
cont>
cont> repeat
cont>     select count(*) into :p
cont>     from employees
cont>     where employee_id = '00000';
cont>     set :v = :v + :p;
cont> until :v > 1000
cont> end repeat;
cont> end;
%RDB-E-NOT_VALID, validation on field P caused operation to fail
```

5.2.14 RMU/SHOW STATISTICS Active User Stall Messages Sorted by Process ID

The RMU/SHOW STATISTICS "Active User Stall Messages" display now includes the ability to sort the list of database users by process ID (OpenVMS PID). The Config option on the horizontal menu at the bottom of the screen can be used to control how the information is to be sorted. By default, the display is unsorted.

5.2.15 RMU /REPAIR /INITIALIZE ONLY_LAREA_TYPE Keyword

This note was inadvertently left out of the Oracle Rdb Release 7.1.0 Release Notes.

A new ONLY_LAREA_TYPE keyword has been added to the RMU /REPAIR /INITIALIZE qualifier. This keyword, along with the /NOSPAM and /NOABM qualifiers, allows only the logical area " type " field to be updated in the AIP (area inventory pages). Avoiding SPAM page updates significantly improves performance of this operation.

The RMU /UNLOAD /AFTER_JOURNAL and RMU /SHOW STATISTICS commands use the on−disk area inventory pages (AIPs) to determine the appropriate "type" of each logical area. However, this logical area information in the AIP is generally unknown for logical areas created prior to Oracle Rdb Release 7.0.1. If the RMU /UNLOAD /AFTER_JOURNAL command cannot determine the logical area type for one or more AIP entries, a warning message is displayed for each such area and may ultimately return logical dbkeys with a "0" (zero) area number for records stored in mixed format storage areas.

In order to update the on disk logical area "type" in the AIP, the RMU /REPAIR utility must be used. The /INITIALIZE = LAREA_PARAMETERS =optionfile qualifier can be used with the /TYPE qualifier. For example, to repair the EMPLOYEES table of the MF_PERSONNEL database, you would create an options file that contains the following line:

```
EMPLOYEES /TYPE=TABLE
```

For partitioned logical areas, the /AREA=name qualifier can be used to identify the specific storage areas that are to be updated. For example, to repair the EMPLOYEES table of the MF_PERSONNEL database for the EMPID_OVER storage area only, you would create an options file that contains the following line:

```
EMPLOYEES /AREA=EMPID_OVER /TYPE=TABLE
```

The /TYPE qualifier specifies the type of a logical area. The following keywords are allowed:

- TABLE − Specifies that the logical area is a data table. This would be a table created using the SQL "CREATE TABLE" syntax.
- B−TREE − Specifies that the logical area is a b−tree index. This would be an index created using the SQL "CREATE INDEX TYPE IS SORTED" syntax.
- HASH − Specifies that the logical area is a hash index. This would be an index created using the SQL "CREATE INDEX TYPE IS HASHED" syntax.
- SYSTEM − Specifies that the logical area is a system record which is used to identify hash buckets. Users cannot explicitly create these types of logical areas. This type should NOT be used for the RDB$SYSTEM logical areas. This type does NOT identify system relations.
- BLOB − Specifies that the logical area is a blob (segmented string; list of byte varying) repository.

There is no explicit error checking of the "type" specified for a logical area. However, an incorrect type may cause the RMU /UNLOAD /AFTER_JOURNAL command to be unable to correctly return valid logical dbkeys.

The ONLY_LAREA_TYPE keyword can be specified along with the /NOSPAM and /NOABM qualifiers to cause *only* the logical area type field to be updated in the area inventory pages. All other actions specified in the options file are ignored when ONLY_LAREA_TYPE is specified. By updating only the logical area type in the AIP entries and not the SPAM pages, the RMU /REPAIR operation can be considerably faster.

5.2.16 RMU/SHOW STATISTICS Cluster Data Collection Performance Enhancement

The RMU /SHOW STATISTICS Utility has been enhanced to perform "asynchronous" data gathering when statistics are being displayed cluster−wide. Previously, a request for statistics was sent to the remote statistics server and then the response was received synchronously. This was repeated for each node being monitored at each data refresh cycle.

Now, the requests for information are sent to all nodes at once and then the replies are read as they become available. This reduces some of the the delay associated with gathering statistics from multiple nodes in a cluster.

5.2.17 RMU Extract has Enhanced Extract of Conditional Expressions

This release of Oracle Rdb now includes support for the new ABS function by RMU Extract. RMU Extract decodes case expressions into ABS (absolute value) functions.

ABS (a) is equivalent to:

```
CASE
  WHEN a < 0 THEN −a
  ELSE a
END
```

In addition, similar forms of CASE expressions are also converted to ABS.

```
CASE
  WHEN a <= 0 THEN −a
  ELSE a
END
```

and

```
CASE
  WHEN a > 0 THEN a
  ELSE −a
END
```

and

```
CASE
  WHEN a >= 0 THEN a
  ELSE −a
END
```

It is possible that RMU Extract will change existing CASE expressions into this more compact syntax, even if it was not originally coded as an ABS function call.

Chapter 6
Documentation Corrections, Additions and Changes

This chapter provides corrections for documentation errors and omissions.

6.1 Documentation Corrections

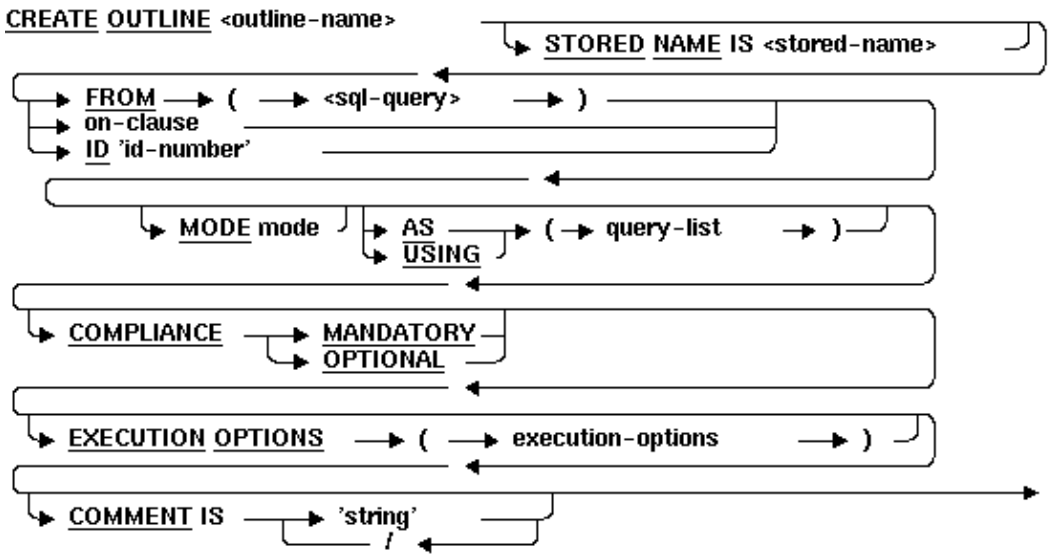
6.1.1 CREATE OUTLINE Supports Trigger, Constraint, Column and View Outlines

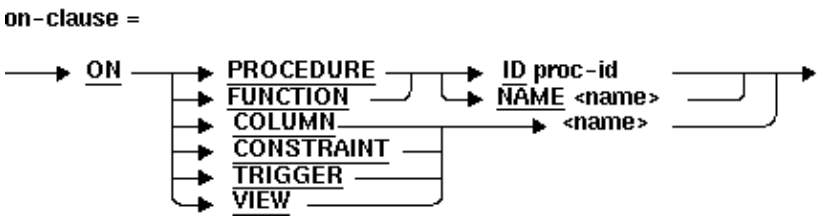
The syntax diagram for the following note was incorrect in the original documentation about it which was Section 1.3.5 in the Oracle Rdb Release 7.1.0 Release Notes.

CREATE OUTLINE now supports direct outline creation for TRIGGERS and CONSTRAINTS, and partial outlines for column expressions (COMPUTED BY and AUTOMATIC), and VIEW definitions.

The CREATE OUTLINE syntax has been enhanced to support the referencing of views, constraints, triggers and columns. The name of the outline in these cases should match the name of the object so that Oracle Rdb may locate the outline definition at runtime.

FORMAT





USAGE NOTES

- When CREATE OUTLINE ... ON TRIGGER is used then an outline for just the first compound trigger action is created. In a future release, outlines for subsequent actions will be supported.
- Partial outlines for view definitions may not be suitable for use in queries without providing more details in the outline. This is shown in a later example.

- CREATE OUTLINE ... ON COLUMN must reference a computed column, such as a table COMPUTED BY, AUTOMATIC or view column that contains select expressions. The CREATE will fail if no select expression is available.

The following example shows the outline created for the CURRENT_JOB view. Note that the access path for JOB_HISTORY defaults to SEQUENTIAL and therefore is not the best choice for this view. This occurs because the view normally queries with an EMPLOYEE_ID specified, which would cause the optimizer to choose index access for the JOB_HISTORY table.

```
SQL> create outline CURRENT_JOB on view CURRENT_JOB;
SQL> show outline CURRENT_JOB
     CURRENT_JOB
 Source:

-- Rdb Generated Outline : 16-MAY-2001 15:11
create outline CURRENT_JOB
-- On view CURRENT_JOB
id '9C6D98DAAF09A3E1796F7D345399028B'
mode 0
as (
  query (
-- View
    subquery (
      JOB_HISTORY 0   access path sequential
        join by cross to
      EMPLOYEES 1     access path index       EMPLOYEES_HASH
      )
    )
  )
compliance optional     ;
```

This alternate definition includes an index on JOB_HISTORY.

```
SQL> create outline CURRENT_JOB
cont>     on view CURRENT_JOB
cont> mode 0
cont> as (
cont>   query (
cont> -- View
cont>     subquery (
cont>       JOB_HISTORY 0   access path index  JH_EMPLOYEE_ID
cont>         join by cross to
cont>       EMPLOYEES 1     access path index  EMPLOYEES_HASH
cont>       )
cont>     )
cont>   )
cont> compliance optional
cont> comment is 'qo for view CURRENT_JOB';
```

The following query shows the results when applying this query outline. The table RETIRED_EMPLOYEES, as the name implies, contains all retired employees. Therefore, there should be no jobs assigned to these employees and the query should return zero rows.

```
SQL> -- should return no rows, since the employee retired and
SQL> -- there is no current job
SQL> set flags 'strategy';
SQL> select EMPLOYEE_ID
cont>  from CURRENT_JOB cj
```

```
cont>       inner join RETIRED_EMPLOYEES re
cont>       using (EMPLOYEE_ID)
cont>  where EMPLOYEE_ID = '00164';
~S: Outline "CURRENT_JOB" used
Cross block of 2 entries
  Cross block entry 1
    Index only retrieval of relation RETIRED_EMPLOYEES
      Index name  RE_EMPLOYEE_ID [1:1]
  Cross block entry 2
    Cross block of 2 entries
      Cross block entry 1
        Conjunct
        Leaf#01 FFirst JOB_HISTORY Card=274
          BgrNdx1 JH_EMPLOYEE_ID [1:1] Bool Fan=17
      Cross block entry 2
        Conjunct        Index only retrieval of relation EMPLOYEES
          Index name  EMPLOYEES_HASH [1:1]       Direct lookup
0 rows selected
SQL>
```

Note that the query outline CURRENT_JOB is reported as being used.

6.1.2 New RMU/BACKUP Storage Area Assignment With Thread Pools

This is to clarify how storage areas are assigned to disk and tape devices using the new RMU/BACKUP THREAD POOL and BACKUP TO MULTIPLE DISK DEVICES features introduced in Oracle Rdb Release 7.1.

For the case of backup to multiple disk devices using thread pools, the algorithm used by RMU/BACKUP to assign threads is to calculate the size of each area as the product of the page length in bytes times the highest page number used (maximum page number) for that area. The area sizes are then sorted by descending size and ascending device name. For internal processing reasons, the system area is placed as the first area in the first thread. Each of the remaining areas is added to whichever thread has the lowest byte count. In this way, the calculated area sizes are balanced between the threads.

For tape devices, the same algorithm is used but the areas are partitioned among writer threads, not disk devices.

The partitioning for backup to multiple disk devices is done by disk device, not by output thread, because there will typically be more disk devices than output threads, and an area can not span a device.

6.1.3 DROP INDEX Now an Online Table Operation

The example for the following note was in error in the original documentation.

DROP INDEX can now be used when other users are processing the table on which the index is defined. This requires that the index has previously been disabled with the ALTER INDEX ... MAINTENANCE IS DISABLED statement.

Once maintenance is disabled for an index, it is no longer used by queries on the table. For example, it is not used for retrieval and it is not updated by INSERT, DELETE or UPDATE statements. Therefore, with this release, Rdb has relaxed the requirement of EXCLUSIVE table access for DROP INDEX.

Oracle recommends that the DROP INDEX statement immediately be followed by a COMMIT statement so that all locks on the system metadata be released. Otherwise, access to this and other tables may be stalled waiting for rows locked in the tables RDB$INDICES, RDB$INDEX_SEGMENTS, RDB$STORAGE_MAPS, and RDB$STORAGE_MAP_AREAS.

This change benefits very large databases (VLDB) which have the need to drop indices stored in MIXED format storage areas on large cardinality tables. These indices may take several hours to erase, which in previous versions required taking the table offline from normal processing until the DROP INDEX completed.

Note that indices stored in UNIFORM format storage areas do not take long to DROP due to optimizations which can be made for UNIFORM areas.

```
-- Disable the index maintenance.  This requires exclusive access to the
-- table, but takes a very short time.  This should be done during normal
-- offline maintenance
--
set transaction read write;
alter index TRANSACTION_POSTING_INDEX
    maintenance is disabled;
commit;

-- Once disabled the index can be dropped at any time
--
set transaction read write;
drop index TRANSACTION_POSTING_INDEX;

commit;
```

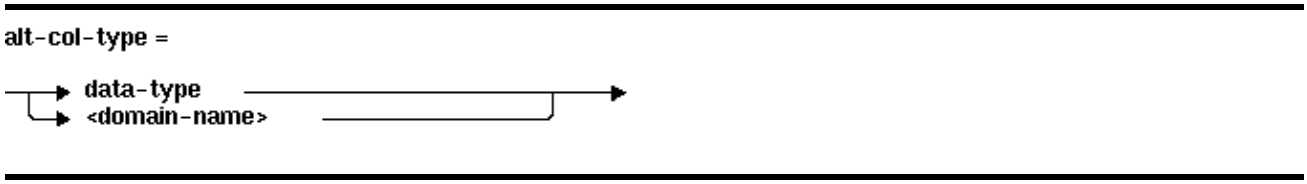
Please note that DROP INDEX will write before image data to the snapshot files (.SNP) if the transaction is started in a mode such as SHARED or PROTECTED. Snapshots can be disabled on the database to avoid the excessive snapshot file I/O during concurrent DROP INDEX operations. Naturally, this may not be possible under normal workloads.

6.1.4 AUTOMATIC Clause Not Supported in ALTER TABLE ... ALTER COLUMN

Bug 2170476

The ALTER TABLE description in the New and Changed Features Manual for Oracle Rdb 7.1 includes a misleading syntax diagram. The alt−col−type diagram includes the AUTOMATIC clause as a possible alternate when altering an existing column using the ALTER COLUMN clause. This functionality is currently not supported by Oracle Rdb.

The revised syntax is:



6.1.5 RDM$BIND_LOCK_TIMEOUT_INTERVAL Overrides the Database Parameter

Bug 2203700

When starting a transaction, there are three different values that are used to determine the lock timeout interval for that transaction. Those values are:

1. The value specified in the SET TRANSACTION statement
2. The value stored in the database as specified in CREATE or ALTER DATABASE
3. The value of the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL

The timeout interval for a transaction is the smaller of the value specified in the SET TRANSACTION statement and the value specified in CREATE DATABASE. However, if the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL is defined, the value of this logical name overrides the value specified in CREATE DATABASE.

The description of how these three values interact, found in several different parts of the Rdb documentation set, is incorrect and will be replaced by the description above.

The lock timeout value in the database can be dynamically modified from the Locking Dashboard in RMU/SHOW STATISTICS. The Per−Process Locking Dashboard can be used to dynamically override the logical name RDM$BIND_LOCK_TIMEOUT_INTERVAL for one or more processes.

6.1.6 New Request Options for RDO, RDBPRE and RDB$INTERPRET

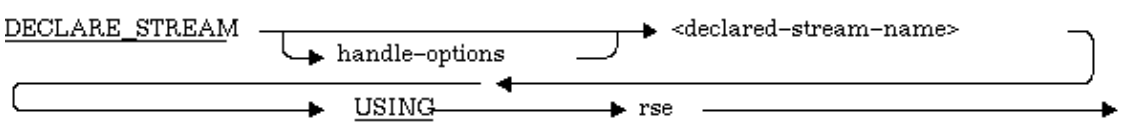
This release note was included in the V70A Release Notes but had gotten dropped somewhere along the line.

For this release of Rdb, two new keywords have been added to the handle−options for the DECLARE_STREAM, the START_STREAM (undeclared format) and FOR loop statements. These changes have been made to RDBPRE, RDO and RDB$INTERPRET at the request of several RDO customers.

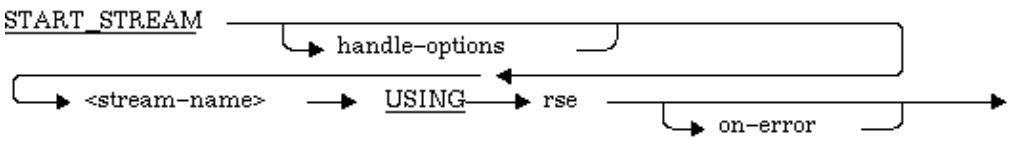
In prior releases, the handle−options could not be specified in interactive RDO or RDB$INTERPRET. This has changed in Rdb7 but these allowed options will be limited to MODIFY and PROTECTED keywords. For RDBPRE, all options listed will be supported. These option names were chosen to be existing keywords to avoid adding any new keywords to the RDO language.

The altered statements are shown in Example 5−1, Example 5−2 and Example 5−3.

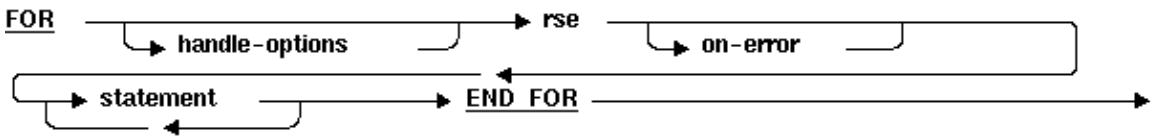
Example 5−1 DECLARE_STREAM Format



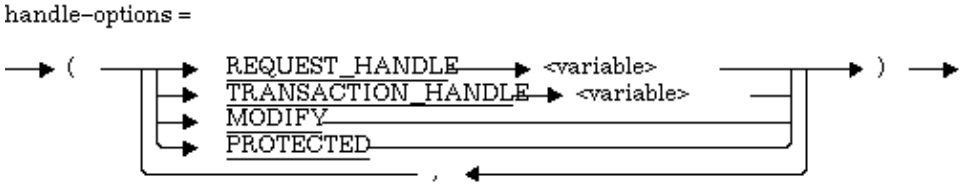
Example 5−2 START_STREAM Format



Example 5−3 FOR Format



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



The following options are available for HANDLE−OPTIONS:

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

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

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

RDMS$BIND_WORK_VM and RDMS$BIND_WORK_FILE.
This option is valid for RDBPRE, RDB$INTERPRET, and interactive RDO. This option is not currently available for RDML.
The following example creates a record stream in a BASIC program using Callable RDO:

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

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

RDMS_STATUS = RDB$INTERPRET ('FETCH EMP')

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

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

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

6.1.7 Missing Descriptions of RDB$FLAGS from HELP File

The HELP file for Oracle Rdb7 describes the system tables for Oracle Rdb and was missing these updated descriptions of the RDB$FLAGS column for several tables.

Table 6−1 Changed Columns for RDB$INDICES Table

Column Name	Data Type	Domain Name	Comments
RDB$FLAGS	Integer	RDB$FLAGS	A bit mask where the bits have the following meaning when set:
			Bit 0: This index is of type HASHED.
			Bit 1: This index uses the MAPPING VALUES clause to compress integer value ranges.
			Bit 2: If this is a HASHED index then it is of type ORDERED. If clear this indicates the index is of type SCATTERED.
			Bit 3: Reserved for future use.
			Bit 4: This index has run length compression enabled (ENABLE COMPRESSION).
			Bit 5: This index is no longer used (MAINTENANCE IS DISABLED).
			Bit 6 through 10: Reserved for future use.
			Bit 11: This index has duplicates compressed (DUPLICATES ARE COMPRESSED).
			Bit 12: This index is of type SORTED RANKED.

Bits 13 through 31: Reserved for future use.

Table 6−2 Changed Columns for RDB$RELATIONS Table

Column Name	Data Type	Domain Name	Comments
RDB$FLAGS	Integer	RDB$FLAGS	A bit mask where the bits have the following meaning when set:
			Bit 0: This relation is a view.
			Bit 1: This relation is not compressed.
			Bit 2: The SQL clause, WITH CHECK OPTION, is used in this view definition.
			Bit 3: Indicates a special internal system relation.
			Bit 4: This view is not an ANSI updatable view.
			Bit 5: This is an imported table in the Distributed Option for Rdb catalog.
			Bit 6: This is a passthru table in the Distributed Option for Rdb catalog.
			Bit 7: This is a partitioned view in the Distributed Option for Rdb catalog.
			Bit 8: This table has compression defined by the storage map. When set Bit 1 in this bit mask is ignored.
			Bit 9: This is a temporary table.
			Bit 10: When bit 9 is set this is a global temporary table, when clear it indicates a local temporary table.
			Bit 11: When bit 9 is set this indicates that the rows in the temporary table should be deleted upon COMMIT.
			Bit 12: Reserved for future use.
			Bit 13: A table (via a computed by column) or view references a local temporary table.
			Bit 14: Reserved for future use.
			Bit 15: This is a system table with a special storage map.
			Bits 16 through 31: Reserved for future use.

Table 6−3 Changed Columns for RDB$STORAGE_MAPS Table

Column Name	Data Type	Domain Name	Comments
RDB$FLAGS	Integer	RDB$FLAGS	A bit mask where the bits have the following meaning when set:
			Bit 0: This table or index is mapped to page format MIXED areas.
			Bit 1: This partition is not compressed.
			Bit 2: This is a strictly partitioned storage map, the partitioning columns become read only for UPDATE.

Bit 3 through 31: Reserved for future use.

6.2 Address and Phone Number Correction for Documentation

In release 7.0 or earlier documentation, the address and fax phone number listed on the Send Us Your Comments page are incorrect. The correct information is:

```
FAX –– 603.897.3825
Oracle Corporation
One Oracle Drive
Nashua, NH 03062–2804
USA
```

6.3 Online Document Format and Ordering Information

You can view the documentation in Adobe Acrobat format using the Acrobat Reader, which allows anyone to view, navigate, and print documents in the Adobe Portable Document Format (PDF). See http://www.adobe.com for information about obtaining a free copy of Acrobat Reader and for information on supported platforms.

The Oracle Rdb documentation in Adobe Acrobat format is available on MetaLink:

```
Top Tech Docs\Oracle Rdb\Documentation\<bookname>
```

In North America, printed documentation is available for sale in the Oracle Store at:

```
http://oraclestore.oracle.com/
```

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from:

```
http://www.oraclebookshop.com/
```

Other customers can contact their Oracle representative to purchase printed documentation.

6.4 New and Changed Features in Oracle Rdb Release 7.1

This section provides information about late−breaking new features or information that is missing or changed since the Oracle Rdb New and Changed Features for Oracle Rdb manual was published.

6.4.1 PERSONA is Supported in Oracle SQL/Services

In the "New and Changed Features for Oracle Rdb" Manual under the section "ALTER DATABASE Statement" is a note stating that impersonation is not supported in Oracle SQL/Services. This is incorrect. There was a problem in the first release of Oracle Rdb 7.1 (7.1.0) whereby impersonation through Oracle SQL/Services failed. This problem is resolved in Oracle Rdb Release 7.1.0.1.

6.4.2 NEXTVAL and CURRVAL Pseudocolumns Can Be Delimited Identifiers

The Oracle Rdb New and Changed Features for Oracle Rdb manual describes SEQUENCES but does not mention that the special pseudocolumns NEXTVAL and CURRVAL can be delimited. All uppercase and lowercase variations of these keywords are accepted and assumed to be equivalent to these uppercase keywords.

The following example shows that any case is accepted:

```
SQL> set dialect 'sql92';
SQL> create sequence dept_id;
SQL> select dept_id.nextval from rdb$database;
                    1
1 row selected
SQL> select "DEPT_ID".currval from rdb$database;
                    1
1 row selected
SQL> select "DEPT_ID"."CURRVAL" from rdb$database;
                    1
1 row selected
SQL> select "DEPT_ID"."nextval" from rdb$database;
                    2
1 row selected
SQL> select "DEPT_ID"."CuRrVaL" from rdb$database;
                    2
1 row selected
```

6.4.3 Only=select_list Qualifier for the RMU Dump After_Journal Command

The Oracle Rdb New and Changed Features for Oracle Rdb manual documents the First=select_list and Last=select_list qualifiers for the RMU Dump After_Journal command. Inadvertently missed was the Only=select_list qualifier.

The First, Last, and Only qualifiers have been added because the Start and End qualifiers are difficult to use since users seldom know, nor can they determine, the AIJ record number in advance of using the RMU Dump After_Journal command.

The select_list clause of these qualifiers consists of a list of one or more of the following keywords:

- TSN=tsn
  Specifies the first, last, or specific TSN in the AIJ journal using the standard [n:]m TSN format.
- TID=tid
  Specifies the first, last or specific TID in the AIJ journal.
- RECORD=record
  Specifies the first or last record in the AIJ journal. This is the same as the existing Start and End qualifiers (which are still supported, but deprecated). This keyword cannot be used with the Only qualifier.
- BLOCK=block#
  Specifies the first or last block in the AIJ journal. This keyword cannot be used with the Only qualifier.
- TIME=date_time
  Specifies the first or last date/time in the AIJ journal using the standard date/time format. This keyword cannot be used with the Only qualifier.

The First, Last, and Only qualifiers are optional. You may specify any or none of them.

The keywords specified for the First qualifier can differ from the keywords specified for the other qualifiers.

For example, to start the dump from the fifth block of the AIJ journal, you would use the following command:

```
RMU/DUMP/AFTER_JOURNAL /FIRST=(BLOCK=5) MF_PERSONNEL.AIJ
```

To start the dump from block 100 or TSN 52, whichever occurs first, you would use the following command:

```
RMU/DUMP/AFTER_JOURNAL /FIRST=(BLOCK=100,TSN=0:52) MF_PERSONNEL.AIJ
```

When multiple keywords are specified for a qualifier, the first condition being encountered activates the qualifier. In the preceding example, the dump starts when either block 100 or TSN 52 is encountered.

Be careful when searching for TSNs or TIDs as they are not ordered in the AIJ journal. For example, if you want to search for a specific TSN, use the Only qualifier and not the First and Last qualifiers. For example, assume the AIJ journal contains records for TSN 150, 170, and 160 (in that order). If you specify the First=TSN=160 and Last=TSN=160 qualifiers, nothing will be dumped because TSN 170 will match the Last=TSN=160 criteria.

6.5 Oracle Rdb7 and Oracle CODASYL DBMS Guide to Hot Standby Databases

This section provides information that is missing from or changed in V7.0 of the Oracle Rdb7 and Oracle CODASYL DBMS Guide to Hot Standby Databases.

6.5.1 Restrictions Lifted on After−Image Journal Files

The Hot Standby software has been enhanced regarding how it handles after−image journal files. Section 4.2.4 in the Oracle Rdb and Oracle CODASYL DBMS Guide to Hot Standby Databases states the following information:

```
If an after−image journal switchover operation is suspended when
replication operations are occurring, you must back up one or more of
the modified after−image journals to add a new journal file.
```

This restriction has been removed. Now, you can add journal files or use the emergency AIJ feature of Oracle Rdb release 7.0 to automatically add a new journal file. Note the following distinctions between adding an AIJ file and adding an emergency AIJ file:

- You can add an AIJ file to the master database and it will be replicated on the standby database. If replication operations are active, the AIJ file is created on the standby database immediately. If replication operations are not active, the AIJ file is created on the standby database when replication operations are restarted.
- You can add emergency AIJ files anytime. If replication operations are active, the emergency AIJ file is created on the standby database immediately. However, because emergency AIJ files are not journaled, starting replication after you create an emergency AIJ will fail. You cannot start replication operations because the Hot Standby software detects a mismatch in the number of after−image journal files on the master compared to the standby database.
  If an emergency AIJ file is created on the master database when replication operations are not active, you must perform a master database backup and then restore the backup on the standby database. Otherwise, an AIJSIGNATURE error results.

6.5.2 Changes to RMU Replicate After_Journal ... Buffer Command

The behavior of the RMU Replicate After_Journal ... Buffers command has been changed. The Buffers qualifier may be used with either the Configure option or the Start option.

When using local buffers, the AIJ Log Roll−forward Server will use a minimum of 4096 buffers. The value provided to the Buffers qualifier will be accepted but ignored if it is less than 4096. In addition, further parameters will be checked and the number of buffers may be increased if the resulting calculations are greater than the number of buffers specified by the Buffers qualifier. If the database is configured to use more than 4096 AIJ Request Blocks (ARBs), then the number of buffers may be increased to the number of ARBs configured for the database. The LRS ensures that there are at least 10 buffers for every possible storage area in the database. Thus if the total number of storage areas (both used and reserved) multiplied by 10 results in a greater number of buffers, then that number will be used.

When global buffers are used, the number of buffers used by the AIJ Log Roll−forward Server is determined as follows:

- If the Buffers qualifier is omitted and the Online qualifier is specified, then the number of buffers will default to the previously configured value, if any, or 256, whichever is larger.
- If the Buffers qualifier is omitted and the Online qualifier is not specified or the Noonline qualifier is specified, then the number of buffers will default to the maximum number of global buffers allowed per user ("USER LIMIT"), or 256, whichever is larger.
- If the Buffers qualifier is specified then that value must be at least 256, and it may not be greater than the maximum number of global buffers allowed per user ("USER LIMIT").

The Buffer qualifier now enforces a minimum of 256 buffers for the AIJ Log Roll−forward Server. The maximum number of buffers allowed is still 524288 buffers.

6.5.3 Unnecessary Command in the Hot Standby Documentation

There is an unnecessary command documented in the Oracle Rdb and Oracle CODASYL DBMS Guide to Hot Standby Databases manual. The documentation (in Section 2.12 "Step 10: Specify the Network Transport Protocol") says that to use TCP/IP as the network protocol, you must issue the following commands:

```
$ CONFIG UCX AIJSERVER OBJECT
$ UCX SET SERVICE RDMAIJSRV
/PORT=n
/USER_NAME=RDMAIJSERVER
/PROCESS_NAME=RDMAIJSERVER
/FILE=SYS$SYSTEM:rdmaijserver_ucx.com
/LIMIT=nn
```

The first of these commands ($ CONFIG UCX AIJSERVER OBJECT) is unnecessary. You can safely disregard the first line when setting up to use TCP/IP with Hot Standby.

The documentation will be corrected in a future release of Oracle Rdb.

6.5.4 Change in the Way RDMAIJ Server is Set Up in UCX

Starting with Oracle Rdb Release 7.0.2.1, the RDMAIJ image became a varianted image. Therefore, the information in Section 2.12, "Step 10: Specify the Network Transport Protocol," of the Oracle Rdb7 and Oracle CODASYL DBMS Guide to Hot Standby Databases has become outdated with regard to setting up the RDMAIJSERVER object when using UCX as the network transport protocol. The UCX SET SERVICE command is now similar to the following:

```
$ UCX SET SERVICE RDMAIJ −
      /PORT=port_number −
      /USER_NAME=RDMAIJ −
      /PROCESS_NAME=RDMAIJ −
      /FILE=SYS$SYSTEM:RDMAIJSERVER.com −
      /LIMIT=limit
```

For Oracle Rdb multiversion, the UCX SET SERVICE command is similar to the following:

```
$ UCX SET SERVICE RDMAIJ70 −
      /PORT=port_number −
      /USER_NAME=RDMAIJ70 −
```

```
/PROCESS_NAME=RDMAIJ70 -
/FILE=SYS$SYSTEM:RDMAIJSERVER70.com -
/LIMIT=limit
```

The installation procedure for Oracle Rdb creates a user named RDMAIJ(nn) and places a file called RDMAIJSERVER(nn).COM in SYS$SYSTEM. The RMONSTART(nn).COM command procedure will try to enable a service called RDMAIJ(nn) if UCX is installed and running.

Changing the RDMAIJ server to a varianted image does not impact installations using DECNet since the correct DECNet object is created during the Oracle Rdb installation.

6.5.5 CREATE INDEX Operation Supported for Hot Standby

On Page 1−13 of the Oracle Rdb7 and Oracle CODASYL DBMS Guide to Hot Standby Databases, the add new index operation is incorrectly listed as an offline operation not supported by Hot Standby. The CREATE INDEX operation is now fully supported by Hot Standby, as long as the transaction does not span all available AIJ journals, including emergency AIJ journals.

6.6 Oracle Rdb7 for OpenVMS Installation and Configuration Guide

This section provides information that is missing from or changed in V7.0 of the Oracle Rdb7 for OpenVMS Installation and Configuration Guide.

6.6.1 Suggestion to Increase GH_RSRVPGCNT Removed

The Oracle Rdb7 for OpenVMS Installation and Configuration Guide contains a section titled "Installing Oracle Rdb Images as Resident on OpenVMS Alpha". This section includes information about increasing the value of the OpenVMS system parameter GH_RSRVPGCNT when you modify the RMONSTART.COM or SQL$STARTUP.COM procedures to install Oracle Rdb images with the Resident qualifier.

Note that modifying the parameter GH_RSRVPGCNT is only required if the RMONSTART.COM or SQL$STARTUP.COM procedures have been manually modified to install Oracle Rdb images with the Resident qualifier. Furthermore, if the RMONSTART.COM and SQL$STARTUP.COM procedures are executed during the system startup procedure (directly from SYSTARTUP_VMS.COM, for example), there is no need to modify the GH_RSRVPGCNT parameter.

Oracle Corporation recommends that you do not modify the value of the GH_RSRVPGCNT system parameter unless it is absolutely required. Some versions of OpenVMS on some hardware platforms require GH_RSRVPGCNT to be a value of zero in order to ensure the highest level of system performance.

6.6.2 Prerequisite Software

In addition to the software listed in the Oracle Rdb Installation and Configuration Guide and at the url http://www.oracle.com/rdb/product_info/index.html, note that the MACRO compiler and linker from Compaq Computer Corporation are required software in order to install Oracle Rdb on your OpenVMS Alpha system.

6.6.3 Defining the RDBSERVER Logical Name

Sections 4.3.7.1 and 4.3.7.2 in the Oracle Rdb7 for OpenVMS Installation and Configuration Guide provide the following examples for defining the RDBSERVER logical name: *$ DEFINE RDBSERVER SYS$SYSTEM:RDBSERVER70.EXE*

and *$ DEFINE RDBSERVER SYS$SYSTEM:RDBSERVER61.EXE*

These definitions are inconsistent with other command procedures that attempt to reference the RDBSERVERxx.EXE image. Below is one example where the RDBSERVER.COM procedure references SYS$COMMON:<SYSEXE> and SYS$COMMON:[SYSEXE] rather than SYS$SYSTEM.

```
$   if .not. -
       ((f$locate ("SYS$COMMON:<SYSEXE>",rdbserver_image) .ne. log_len) .or. -
        (f$locate ("SYS$COMMON:[SYSEXE]",rdbserver_image) .ne. log_len))
$   then
$       say "''rdbserver_image' is not found in SYS$COMMON:<SYSEXE>"
$       say "RDBSERVER logical is ''rdbserver_image'"
$       exit
$   endif
```

In this case, if the logical name were defined as instructed in the Oracle Rdb7 for OpenVMS Installation and Configuration Guide, the image would not be found.

The correct definition of the logical name is as follows: *DEFINE RDBSERVER SYS$COMMON:<SYSEXE>RDBSERVER70.EXE*

and *DEFINE RDBSERVER SYS$COMMON:<SYSEXE>RDBSERVER61.EXE*

6.7 Guide to Database Design and Definition

This section provides information that is missing from or changed in release 7.0 of the Oracle Rdb7 Guide to Database Design and Definition.

6.7.1 Lock Timeout Interval Logical Incorrect

On Page 7−31 of Section 7.4.8 in the Oracle Rdb7 Guide to Database Design and Definition, the RDM$BIND_LOCK_TIMEOUT logical name is referenced incorrectly. The correct logical name is RDM$BIND_LOCK_TIMEOUT_INTERVAL.

The Oracle Rdb7 Guide to Database Design and Definition will be corrected in a future release.

6.7.2 Example 4−13 and Example 4−14 Are Incorrect

Example 4−13 showing vertical partitioning, and Example 4−14, showing vertical and horizontal partitioning, are incorrect. They should appear as follows:

Example 4−13:

```
SQL> CREATE STORAGE MAP EMPLOYEES_1_MAP
cont>    FOR EMPLOYEES
cont>    ENABLE COMPRESSION
cont>    STORE COLUMNS (EMPLOYEE_ID, LAST_NAME, FIRST_NAME,
cont>                   MIDDLE_INITIAL, STATUS_CODE)
cont>                   DISABLE COMPRESSION
cont>                   IN ACTIVE_AREA
cont>    STORE COLUMNS (ADDRESS_DATA_1, ADDRESS_DATA_2, CITY,
cont>                   STATE, POSTAL_CODE)
cont>                   IN INACTIVE_AREA
cont>    STORE IN OTHER_AREA;
```

Example 4−14:

```
SQL>  CREATE STORAGE MAP EMPLOYEES_1_MAP2
cont>        FOR EMP2
cont>        STORE COLUMNS (EMPLOYEE_ID, LAST_NAME, FIRST_NAME,
cont>                       MIDDLE_INITIAL, STATUS_CODE)
cont>             USING (EMPLOYEE_ID)
cont>             IN ACTIVE_AREA_A WITH LIMIT OF ('00399')
cont>             IN ACTIVE_AREA_B WITH LIMIT OF ('00699')
cont>             OTHERWISE IN ACTIVE_AREA_C
cont>        STORE COLUMNS (ADDRESS_DATA_1, ADDRESS_DATA_2, CITY,
cont>                      STATE, POSTAL_CODE)
cont>             USING (EMPLOYEE_ID)
cont>             IN INACTIVE_AREA_A WITH LIMIT OF ('00399')
cont>             IN INACTIVE_AREA_B WITH LIMIT OF ('00699')
cont>             OTHERWISE IN INACTIVE_AREA_C
cont>        STORE IN OTHER_AREA;
```

6.8 Oracle Rdb7 SQL Reference Manual

This section provides information that is missing from or changed in V7.0 of the Oracle Rdb7 SQL Reference Manual.

6.8.1 Clarification of the DDLDONOTMIX Error Message

The ALTER DATABASE statement performs two classes of functions:

1. Changing the database root structures in the .RDB file
2. Modifying the system metadata in the RDB$SYSTEM storage area.

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

When you mix clauses that fall into both classes, the error message DDLDONOTMIX "the {SQL−syntax} clause can not be used with some ALTER DATABASE clauses" is displayed, and the ALTER DATABASE statement fails. For example:

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

The following clauses may be mixed with each other, but may not appear with other clauses such as ADD STORAGE AREA or ADD CACHE:

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

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

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

6.8.2 Node Specification Allowed on Root FILENAME Clauses

In previous releases of the Oracle Rdb SQL Reference Manual, it was not made clear that a node specification may only be specified for the root FILENAME clause of the ALTER DATABASE, CREATE DATABASE, EXPORT DATABASE, and IMPORT DATABASE statements.

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

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

Usage notes reflecting this restriction for these clauses will appear in a future release of the Oracle Rdb SQL Reference Manual.

6.8.3 Incorrect Syntax Shown for Routine−Clause of the CREATE MODULE Statement

The Oracle Rdb7 SQL Reference Manual incorrectly showed that a simple−statement could be specified for the routine−clause of the CREATE MODULE statement. You can specify a compound−statement and compound−use−statement for the routine−clause only of the CREATE MODULE statement.

This correction appears in the Oracle Rdb New and Changed Features for Oracle Rdb manual and will appear in a future release of the Oracle Rdb7 SQL Reference Manual.

6.8.4 Omitted SET Statements

The following SET statements and language options were omitted from the Oracle Rdb7 SQL Reference Manual.

6.8.4.1 QUIET COMMIT

The following QUIET COMMIT options were omitted from the documentation:

```
Interactive and dynamic SET QUIET COMMIT statement
SQL
Module Header           QUIET COMMIT option
SQL Module Language     /QUIET_COMMIT and /NOQUIET_COMMIT qualifiers
SQL Precompiler         /SQLOPTIONS=QUIET_COMMIT and
                        /SQLOPTIONS=NOQUIET_COMMIT options
```

These options control the behavior of the COMMIT and ROLLBACK statements in cases where there is no active transaction.

By default, if there is no active transaction, SQL will raise an error when COMMIT or ROLLBACK is executed. This default is retained for backward compatibility for applications that wish to detect the situation. If QUIET COMMIT is set to ON, a COMMIT or ROLLBACK executes successfully when there is no active transaction.

Within a compound statement, the COMMIT and ROLLBACK statements are ignored.

In interactive or dynamic SQL, the SET statement can be used to disable or enable error reporting for COMMIT and ROLLBACK when no transaction is active. The parameter to the SET command is a string literal or host variable containing the keyword ON or OFF. For example:

```
SQL> COMMIT;
%SQL-F-NO_TXNOUT, No transaction outstanding
SQL> ROLLBACK;
%SQL-F-NO_TXNOUT, No transaction outstanding
SQL> SET QUIET COMMIT 'on';
SQL> ROLLBACK;
SQL> COMMIT;
SQL> SET QUIET COMMIT 'off';
SQL> COMMIT;
%SQL-F-NO_TXNOUT, No transaction outstanding
```

In the SQL module language or precompiler header, the QUIET COMMIT option can be used to disable or enable error reporting for COMMIT and ROLLBACK when no transaction is active. The keyword ON or OFF must be used to enable or disable this feature. The following example enables QUIET COMMIT so that no error is reported if a COMMIT is executed when no transaction is active:

```
MODULE TXN_CONTROL
LANGUAGE BASIC
PARAMETER COLONS
QUIET COMMIT ON

PROCEDURE S_TXN (SQLCODE);
SET TRANSACTION READ WRITE;

PROCEDURE C_TXN (SQLCODE);
COMMIT;
```

6.8.4.2 COMPOUND TRANSACTIONS

The SET COMPOUND TRANSACTIONS statement (for interactive and dynamic SQL) and the module header option, COMPOUND TRANSACTIONS, controls the SQL behavior for starting default transactions for compound statements.

By default, if there is no current transaction, SQL will start a transaction before executing a compound statement or stored procedure. However, this may conflict with the actions within the procedure or may start a transaction for no reason if the procedure body does not perform database access. This default is retained for backward compatibility for applications which may expect a transaction to be started for the procedure.

If COMPOUND TRANSACTIONS is set to EXTERNAL, SQL starts a transaction before executing the procedure. Otherwise, if it is set to INTERNAL, it allows the procedure to start a transaction as required by the procedure execution.

In interactive or dynamic SQL, the following SET command can be used to disable or enable transactions starting by the SQL interface. The parameter to the SET command is a string literal or host variable containing the keyword 'INTERNAL' or 'EXTERNAL'.

```
SQL> SET COMPOUND TRANSACTIONS 'internal';
SQL> CALL START_TXN_AND_COMMIT ();
SQL> SET COMPOUND TRANSACTIONS 'external';
SQL> CALL UPDATE_EMPLOYEES (...);
```

In the SQL module language or precompiler header, the COMPOUND TRANSACTIONS option can be used to disable or enable starting a transaction for procedures. The keyword INTERNAL or EXTERNAL must be used to enable or disable this feature.

```
MODULE TXN_CONTROL
LANGUAGE BASIC
PARAMETER COLONS
COMPOUND TRANSACTIONS INTERNAL

PROCEDURE S_TXN (SQLCODE);
BEGIN
SET TRANSACTION READ WRITE;
END;

PROCEDURE C_TXN (SQLCODE);
BEGIN
COMMIT;
END;
```

6.8.5 Size Limit for Indexes with Keys Using Collating Sequences

When a column is defined with a collating sequence, the index key is specially encoded to incorporate the correct collating information. This special encoding takes more space than keys encoded for ASCII (which is the default when no collating sequence is used). Therefore, the encoded string uses more than the customary one byte per character of space within the index. This is true for all versions of Oracle Rdb which support collating sequences.

For all collating sequences, except Norwegian, the space required is approximately 9 bytes for every 8 characters. Therefore, a CHAR (24) column will require approximately 27 bytes to store. For Norwegian collating sequences, the space required is approximately 10 bytes for every 8 characters.

The space required for encoding the string must be taken into account when calculating the size of an index key against the limit of 255 bytes. Suppose a column defined with a collating sequence of GERMAN was used in an index. The length of that column is limited to a maximum of 225 characters because the key will be encoded in 254 bytes.

The following example demonstrates how a 233 character column, defined with a German collating sequence and included in an index, exceeds the index size limit of 255 bytes, even though the column is defined as less than 255 characters in length.

```
SQL> CREATE DATABASE
cont>      FILENAME 'testdb.rdb'
```

```
cont>      COLLATING SEQUENCE GERMAN GERMAN;
SQL> CREATE TABLE employee_info
cont>      (emp_name CHAR (233));
SQL> CREATE INDEX emp_name_idx
cont>      ON employee_info (
cont>      emp_name     ASC)
cont>      TYPE IS SORTED;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-INDTOOBIG, requested index is too big
```

6.8.6 Clarification of SET FLAGS Option DATABASE_PARAMETERS

The Oracle Rdb7 SQL Reference Manual described the option DATABASE_PARAMETERS in table 7−6 in the SET FLAGS section. However, this keyword generates output only during ATTACH to the database which happens prior to the SET FLAGS statement executing.

This option is therefore only useful when used with the RDMS$SET_FLAGS logical name which provides similar functionality.

```
$ define RDMS$SET_FLAGS "database_parameters"
$ sql$
SQL> Attach 'File db$:scratch';
 ATTACH #1, Database BLUGUM$DKA300:[SMITHI.DATABASES.V70]SCRATCH.RDB;1
~P Database Parameter Buffer (version=2, len=79)
0000 (00000) RDB$K_DPB_VERSION2
0001 (00001) RDB$K_FACILITY_ALL
0002 (00002) RDB$K_DPB2_IMAGE_NAME "NODE::DISK:[DIR]SQL$70.EXE;1"
0040 (00064) RDB$K_FACILITY_ALL
0041 (00065) RDB$K_DPB2_DBKEY_SCOPE (Transaction)
0045 (00069) RDB$K_FACILITY_ALL
0046 (00070) RDB$K_DPB2_REQUEST_SCOPE (Attach)
004A (00074) RDB$K_FACILITY_RDB_VMS
004B (00075) RDB$K_DPB2_CDD_MAINTAINED (No)
 RDMS$BIND_WORK_FILE = "DISK:[DIR]RDMSTTBL$UEOU3LQ0RV2.TMP;" (Visible = 0)
SQL> Exit
 DETACH #1
```

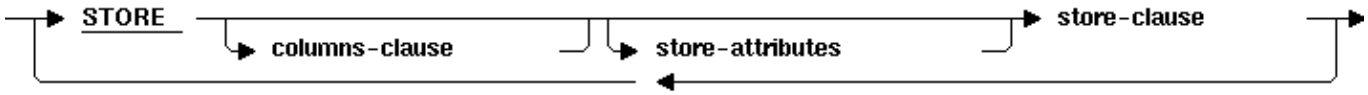
6.8.7 Incorrect Syntax for CREATE STORAGE MAP Statement

The main diagram of the CREATE STORAGE MAP statement incorrectly shows the partition−clause as required syntax. The partition−clause is not a required clause.

The partition−clause diagram of the CREATE STORAGE MAP statement incorrectly indicated that the STORE keyword was not repeated. When creating a vertically partitioned table you must repeat the STORE keyword for each partition.

FORMAT

partition-clause =



When creating a vertical record partition, the last STORE clause cannot contain the COLUMNS clause. If you attempt to include the COLUMNS clause on the last STORE clause, you will an error similar to the following:

```
%SQL-F-VRP_ILLEGAL_STO, Storage Map "EMPLOYEES_MAP2" specified STORE COLUMNS
after a STORE
```

The following example shows the correct syntax for creating a storage map with horizontal and vertical partitions:

```
SQL> CREATE STORAGE MAP employees_map2
cont>     FOR employees2
cont> --
cont> -- Store the primary information horizontally partitioned
cont> -- across the areas EMPIDS_LOW, EMPIDS_MID and EMPIDS_OVER.
cont> -- Disable compression because these columns are accessed often.
cont> --
cont>     STORE
cont>         COLUMNS (employee_id, last_name,
cont>                  first_name, middle_initial)
cont>     VERTICAL PARTITION volatile_columns
cont>         DISABLE COMPRESSION
cont>         USING (employee_id)
cont>             IN empids_low
cont>             (PARTITION id_low)
cont>                 WITH LIMIT OF (`00200')
cont>             IN empids_mid
cont>             (PARTITION id_mid)
cont>                 WITH LIMIT OF (`00400')
cont>             OTHERWISE IN empids_over
cont>             (partition id_ovr)
cont> --
cont> -- Place all the address information in EMP_INFO.
cont> -- Make sure these character columns are compressed.
cont> --
cont>     STORE
cont>         COLUMNS (address_data_1, address_data_2, city, state,
cont>                  postal_code)
cont>         ENABLE COMPRESSION
cont>         IN emp_info
cont> --
cont> -- The remaining columns get written randomly over these area.
cont> --
cont>     STORE
cont>         ENABLE COMPRESSION
cont>         RANDOMLY ACROSS (salary_history, jobs);
```

Refer to Oracle Rdb New and Changed Features for Oracle Rdb for the full syntax of the CREATE STORAGE MAP statement. The Oracle Rdb7 SQL Reference Manual will be corrected in a future release.

6.8.8 Use of SQL_SQLCA Include File Intended for Host Language File

Use of the SQLCA include files such as the SQL_SQLCA.H file for C, are intended for use with the host language files only. That is, only *.C should be including that file. Precompiled files (*.SC files) should use the EXEC SQL INCLUDE SQLCA embedded SQL command in the declaration section of the module. In this way the precompiler can properly define the structure to be used by the related SQL generated code.

Remember that the SQLCA is always scoped at the module level, unlike the SQLCODE or SQLSTATE variables which may be routine specific.

The following example shows this error:

```
#include <stdio.h>
#include <sql_sqlca.h>
struct SQLCA SQLCA;

int main (void)
{
EXEC SQL EXECUTE IMMEDIATE `show version';
printf ("SQLCODE=%d\n", SQLCA.SQLCODE);
}
$ SQLPRE/CC issues the following error against this program:
%SQL-F-NOSQLCODE, Neither SQLCA, SQLCODE nor SQLSTATE were declared
```

The following example shows correct usage:

```
#include <stdio.h>
#include <sql_sqlca.h>
EXEC SQL INCLUDE SQLCA;

int main (void)
{
EXEC SQL EXECUTE IMMEDIATE `show version';
printf ("SQLCODE=%d\n", SQLCA.SQLCODE);
}
```

6.8.9 Missing Information on Temporary Tables

The following information was inadvertently omitted from the Oracle Rdb7 SQL Reference Manual. (Should be in the Usage Notes for CREATE TEMPORARY TABLE.)

Data for a temporary table is stored in virtual memory, not in a storage area. For journaling purposes, when changes are made to the data in a temporary table such as updates or deletes, recovery space is required to hold before images of deleted and updated rows. This recovery space also requires virtual memory and may result in having to increase Page File Quota and Virtual Page Count on OpenVMS.

A recommended way to reduce memory usage when using temporary tables is to commit transactions which modify temporary table data as soon as possible. Upon commit the additional copies of data are released and available for reuse by Oracle Rdb. This eliminates extra copies of data and therefore reduces virtual memory usage.

See the Oracle Rdb7 Guide to Database Design and Definition for calculating memory usage for temporary tables.

6.9 Oracle RMU Reference Manual, Release 7.0

This section provides information that is missing from or changed in V7.0 of the Oracle RMU Reference Manual.

6.9.1 RMU Unload After_Journal Null Bit Vector Clarification

Each output record from the RMU /UNLOAD /AFTER_JOURNAL command includes a vector (array) of bits. There is one bit for each field in the data record. If a null bit value is 1, the corresponding field is NULL; if a null bit value is 0, the corresponding field is not NULL and contains an actual data value. The contents of a data field that is NULL are not initialized and are not predictable.

The null bit vector begins on a byte boundary. The field RDB$LM_NBV_LEN indicates the number of valid bits (and thus, the number of columns in the table). Any extra bits in the final byte of the vector after the final null bit are unused and the contents are unpredictable.

The following example C program demonstrates one possible way of reading and parsing a binary output file (including the null bit vector) from the RMU /UNLOAD /AFTER_JOURNAL command. This sample program has been tested using Oracle Rdb V7.0.5 and higher and Compaq C V6.2−009 on OpenVMS Alpha V7.2−1. It is meant to be used as a template for writing your own program.

```
/* DATATYPES.C */

#include <stdio.h>
#include <descrip.h>
#include <starlet.h>
#include <string.h>

#pragma member_alignment __save
#pragma nomember_alignment

struct { /* Database key structure */
    unsigned short      lno;    /* line number */
    unsigned int        pno;    /* page number */
    unsigned short      dbid;   /* area number */
    } dbkey;

typedef struct { /* Null bit vector with one bit for each column */
    unsigned            n_tinyint   :1;
    unsigned            n_smallint  :1;
    unsigned            n_integer   :1;
    unsigned            n_bigint    :1;
    unsigned            n_double    :1;
    unsigned            n_real      :1;
    unsigned            n_fixstr    :1;
    unsigned            n_varstr    :1;
    } nbv_t;

struct { /* LogMiner output record structure for table DATATYPES */
    char                rdb$lm_action;
    char                rdb$lm_relation_name [31];
    int                 rdb$lm_record_type;
    short               rdb$lm_data_len;
    short               rdb$lm_nbv_len;
    __int64             rdb$lm_dbk;
    __int64             rdb$lm_start_tad;
```

```
    __int64             rdb$lm_commit_tad;
    __int64             rdb$lm_tsn;
    short               rdb$lm_record_version;
    char                f_tinyint;
    short               f_smallint;
    int                 f_integer;
    __int64             f_bigint;
    double              f_double;
    float               f_real;
    char                f_fixstr[10];
    short               f_varstr_len;   /* length of varchar */
    char                f_varstr[10];   /* data of varchar */
    nbv_t               nbv;
    } lm;

#pragma member_alignment __restore

main ()
{   char timbuf [24];
    struct dsc$descriptor_s dsc = {
        23, DSC$K_DTYPE_T, DSC$K_CLASS_S, timbuf};
    FILE *fp = fopen ("datatypes.dat", "r", "ctx=bin");

    memset (&timbuf, 0, sizeof(timbuf));

    while (fread (&lm, sizeof(lm), 1, fp) != 0)
    {
        printf ("Action     = %c\n",     lm.rdb$lm_action);
        printf ("Table      = %.*s\n",   sizeof(lm.rdb$lm_relation_name),
                                         lm.rdb$lm_relation_name);
        printf ("Type       = %d\n",     lm.rdb$lm_record_type);
        printf ("Data Len   = %d\n",     lm.rdb$lm_data_len);
        printf ("Null Bits  = %d\n",     lm.rdb$lm_nbv_len);

        memcpy (&dbkey, &lm.rdb$lm_dbk, sizeof(lm.rdb$lm_dbk));
        printf ("DBKEY      = %d:%d:%d\n", dbkey.dbid,
                                           dbkey.pno,
                                           dbkey.lno);

        sys$asctim (0, &dsc, &lm.rdb$lm_start_tad, 0);
        printf ("Start TAD  = %s\n", timbuf);

        sys$asctim (0, &dsc, &lm.rdb$lm_commit_tad, 0);
        printf ("Commit TAD = %s\n", timbuf);

        printf ("TSN        = %Ld\n",    lm.rdb$lm_tsn);
        printf ("Version    = %d\n",     lm.rdb$lm_record_version);

        if (lm.nbv.n_tinyint == 0)
                printf ("f_tinyint  = %d\n", lm.f_tinyint);
        else    printf ("f_tinyint  = NULL\n");

        if (lm.nbv.n_smallint == 0)
                printf ("f_smallint = %d\n", lm.f_smallint);
        else    printf ("f_smallint = NULL\n");

        if (lm.nbv.n_integer == 0)
                printf ("f_integer  = %d\n", lm.f_integer);
        else    printf ("f_integer  = NULL\n");

        if (lm.nbv.n_bigint == 0)
```

```
                printf ("f_bigint   = %Ld\n", lm.f_bigint);
        else    printf ("f_bigint   = NULL\n");

        if (lm.nbv.n_double == 0)
                printf ("f_double   = %f\n", lm.f_double);
        else    printf ("f_double   = NULL\n");

        if (lm.nbv.n_real == 0)
                printf ("f_real     = %f\n", lm.f_real);
        else    printf ("f_real     = NULL\n");

        if (lm.nbv.n_fixstr == 0)
                printf ("f_fixstr   = %.*s\n", sizeof (lm.f_fixstr),
                                                    lm.f_fixstr);
        else    printf ("f_fixstr   = NULL\n");

        if (lm.nbv.n_varstr == 0)
                printf ("f_varstr   = %.*s\n", lm.f_varstr_len, lm.f_varstr);
        else    printf ("f_varstr   = NULL\n");

        printf ("\n");
    }
}
```

Example sequence of commands to create a table, unload the data and display the contents with this program:

```
SQL> ATTACH 'FILE MF_PERSONNEL';
SQL> CREATE TABLE DATATYPES (
       F_TINYINT TINYINT
      ,F_SMALLINT SMALLINT
      ,F_INTEGER INTEGER
      ,F_BIGINT BIGINT
      ,F_DOUBLE DOUBLE PRECISION
      ,F_REAL REAL
      ,F_FIXSTR CHAR (10)
      ,F_VARSTR VARCHAR (10));
SQL> COMMIT;
SQL> INSERT INTO DATATYPES VALUES (1, NULL, 2, NULL, 3, NULL, 'THIS', NULL);
SQL> INSERT INTO DATATYPES VALUES (NULL, 4, NULL, 5, NULL, 6, NULL, 'THAT');
SQL> COMMIT;
SQL> EXIT;
$ RMU /BACKUP /AFTER_JOURNAL MF_PERSONNEL AIJBCK.AIJ
$ RMU /UNLOAD /AFTER_JOURNAL MF_PERSONNEL AIJBCK.AIJ -
     /TABLE = (NAME=DATATYPES, OUTPUT=DATATYPES.DAT)
$ CC DATATYPES.C
$ LINK DATATYPES.OBJ
$ RUN DATATYPES.EXE
```

6.9.2 New Transaction_Mode Qualifier for Oracle RMU Commands

A new qualifier, Transaction_Mode, has been added to the RMU Copy, Move_Area, Restore, and Restore Only_Root commands. You can use this qualifier to set the allowable transaction modes for the database root file created by these commands. If you are not creating a root file as part of one of these commands, for example, you are restoring an area, attempting to use this qualifier returns a CONFLSWIT error. This qualifier is similar to the SET TRANSACTION MODE clause of the CREATE DATABASE command in

interactive SQL.

The primary use of this qualifier is when you restore a backup file (of the master database) to create a Hot Standby database. Include the Transaction_Mode qualifier on the RMU Restore command when you create the standby database (prior to starting replication operations). Because only read−only transactions are allowed on the standby database, you should use the Transaction_Mode=Read_Only qualifier setting. This setting prevents modifications to the standby database at all times, even when replication operations are not active.

You can specify the following transaction modes for the Transaction_Mode qualifier:

```
All
Current
None
[No]Batch_Update
[No]Read_Only
[No]Exclusive
[No]Exclusive_Read
[No]Exclusive_Write
[No]Protected
[No]Protected_Read
[No]Protected_Write
[No]Shared
[No]Shared_Read
[No]Shared_Write
```

Note that [No] indicates that the value can be negated. For example, the NoExclusive_Write option indicates that exclusive write is not an allowable access mode for this database. If you specify the Shared, Exclusive, or Protected option, Oracle RMU assumes you are referring to both reading and writing in these modes. For example, the Transaction_Mode=Shared option indicates that you want both Shared_Read and Shared_Write as transaction modes. No mode is enabled unless you add that mode to the list or you use the ALL option to enable all modes.

You cannot negate the following three options: All, which enables all transaction modes; None, which disables all transaction modes; and Current, which enables all transaction modes that are set for the source database. If you do not specify the Transaction_Mode qualifier, Oracle RMU uses the transaction modes enabled for the source database.

You can list one qualifier that enables or disables a particular mode followed by another that does the opposite. For example, Transaction_Mode=(NoShared_Write, Shared) is ambiguous because the first value disables Shared_Write access while the second value enables Shared_Write access. Oracle RMU resolves the ambiguities by first enabling all modes that are enabled by the items in the Transaction_Mode list and then disabling those modes that are disabled by items in the Transaction_Mode list. The order of items in the list is irrelevant. In the example discussed, Shared_Read is enabled and Shared_Write is disabled.

The following example shows how to set a newly restored database to allow read−only transactions only. After Oracle RMU executes the command, the database is ready for you to start Hot Standby replication operations.

```
$ RMU/RESTORE/TRANSACTION_MODE=READ_ONLY MF_PERSONNEL.RBF
```

6.9.3 RMU Server After_Journal Stop Command

If database replication is active and you attempt to stop the database AIJ Log Server, Oracle Rdb returns an error. You must stop database replication before attempting to stop the server.

In addition, a new qualifier, Output=filename, has been added to the RMU Server After_Journal Stop command. This optional qualifier allows you to specify the file where the operational log is to be created. The operational log records the transmission and receipt of network messages.

If you do not include a directory specification with the file name, the log file is created in the database root file directory. It is invalid to include a node name as part of the file name specification.

Note that all Hot Standby bugcheck dumps are written to the corresponding bugcheck dump file; bugcheck dumps are not written to the file you specify with the Output qualifier.

6.9.4 Incomplete Description of Protection Qualifier for RMU Backup After_Journal Command

The description of the Protection Qualifier for the RMU Backup After_Journal command is incomplete in the Oracle RMU Reference Manual for Digital UNIX. The complete description is as follows:

The Protection qualifier specifies the system file protection for the backup file produced by the RMU Backup After_Journal command. If you do not specify the Protection qualifier, the default access permissions are −rw−r−−−−− for backups to disk or tape.

Tapes do not allow delete or execute access and the superuser account always has both read and write access to tapes. In addition, a more restrictive class accumulates the access rights of the less restrictive classes.

If you specify the Protection qualifier explicitly, the differences in access permissions applied for backups to tape or disk as noted in the preceding paragraph are applied. Thus, if you specify Protection=(S,O,G:W,W:R), the access permissions on tape becomes rw−rw−r−.

6.9.5 RMU Extract Command Options Qualifier

A documentation error exists in the description of the Options=options−list qualifier of the RMU Extract command. Currently, the documentation states that this qualifier is not applied to output created by the Items=Volume qualifier. This is incorrect. Beginning with 6.1 of Oracle Rdb, the behavior of the Options=options−list qualifier is applied to output created by the Items=Volume qualifier.

6.9.6 RDM$SNAP_QUIET_POINT Logical is Incorrect

On page 2−72 of the Oracle RMU Reference Manual, the reference to the RDM$SNAP_QUIET_POINT logical is incorrect. The correct logical name is RDM$BIND_SNAP_QUIET_POINT.

6.9.7 Using Delta Time with RMU Show Statistics Command

Oracle RMU does not support the use of delta time. However, because the OpenVMS platform does, there is a workaround. You can specify delta time using the following syntax with the RMU Show Statistics command:

```
$ RMU/SHOW STATISTICS/OUTPUT=file-spec/UNTIL=" ' ' f$cvtime ("+7:00") ' "
```

The +7:00 adds 7 hours to the current time.

You can also use "TOMORROW" and "TODAY+n".

This information will be added to the description of the Until qualifier of the RMU Show Statistics command in a future release of the Oracle RMU Reference Manual.

6.10 Oracle Rdb7 Guide to Database Performance and Tuning

The following section provides corrected, clarified, or omitted information for the Oracle Rdb7 Guide to Database Performance and Tuning manual.

6.10.1 Dynamic OR Optimization Formats

In Table C−2 on Page C−7 of the Oracle Rdb7 Guide to Database Performance and Tuning, the dynamic OR optimization format is incorrectly documented as [l:h...]n. The correct formats for Oracle Rdb Release 7.0 and later are [(l:h)n] and [l:h,l2:h2].

6.10.2 Oracle Rdb Logical Names

The Oracle Rdb7 Guide to Database Performance and Tuning contains a table in Chapter 2 summarizing the Oracle Rdb logical names. The information in the following table supersedes the entries for the RDM$BIND_RUJ_ALLOC_BLKCNT and RDM$BIND_RUJ_EXTEND_BLKCNT logical names.

RDM$BIND_RUJ_ALLOC_BLKCNT Allows you to override the default value of the .ruj file. The block count value can be defined between 0 and 2 billion with a default of 127.

RDM$BIND_RUJ_EXTEND_BLKCNT Allows you to pre−extend the .ruj files for each process using a database. The block count value can be defined between 0 and 65535 with a default of 127.

6.10.3 Waiting for Client Lock Message

The Oracle Rdb7 Guide to Database Performance and Tuning contains a section in Chapter 3 that describes the Performance Monitor Stall Messages screen. The section contains a list describing the "Waiting for" messages. The description of the "waiting for client lock" message was missing from the list.

A client lock indicates that an Rdb metadata lock is in use. The term client indicates that Rdb is a client of the Rdb locking services. The metadata locks are used to guarantee memory copies of the metadata (table, index and column definitions) are consistent with the on−disk versions.

The "waiting for client lock" message means the database user is requesting an incompatible locking mode. For example, when trying to drop a table which is in use, the drop operation requests a PROTECTED WRITE lock on the metadata object (such as a table) which is incompatible with the existing PROTECTED READ lock currently used by others of the table.

These metadata locks consist of three longwords. The lock is displayed in text format first, followed by its hexadecimal representation. The text version masks out non−printable characters with a dot (.).

The leftmost value seen in the hexadecimal output contains the id of the object. The id is described below for tables and views, routines, modules, storage map areas, and sequences.

- For tables and views, the id represents the unique value found in the RDB$RELATION_ID column of the RDB$RELATIONS system relation for the given table.

- For routines, the id represents the unique value found in the RDB$ROUTINE_ID column of the RDB$ROUTINES system relation for the given routine.
- For modules, the id represents the unique value found in the RDB$MODULE_ID column of the RDB$MODULES system relation for the given module.
- For storage map areas, the id represents the physical area id. The "waiting for client lock" message on storage map areas is very rare. This may be raised for databases which have been converted from versions prior to Oracle Rdb 5.1.
- For sequences, the id represents the unique value found in the RDB$SEQUENCE_ID column of the RDB$SEQUENCES system relation for the given sequence.

The next value displayed signifies the object type. The following table describes objects and their hexadecimal type values.

Table 6−4 Objects and Their Hexadecimal Type Value

Object	Hexadecimal Value
Tables or views	00000004
Routines	00000006
Modules	00000015
Storage map	0000000E
Sequences	00000019

The last value in the hexadecimal output represents the lock type. The value 55 indicates this is a client lock.

The following example shows a "waiting for client lock" message from a Stall Messages screen:

```
Process.ID Since......   Stall.reason............................ Lock.ID.
46001105:2 10:40:46.38 − waiting for client '........' 0000001900000004000000055
```

To determine the name of the referenced object given the lock ID the following queries can be used based on the object type:

```
SQL>select RDB$RELATION_NAME from RDB$RELATIONS where RDB$RELATION_ID = 25;
SQL>select RDB$MODULE_NAME from RDB$MODULES where RDB$MODULE_ID = 12;
SQL>select RDB$ROUTINE_NAME from RDB$ROUTINES where RDB$ROUTINE_ID = 7;
SQL>select RDB$SEQUENCE_NAME from RDB$SEQUENCES where RDB$SEQUENCE_ID = 2;
```

Because the full client lock output is long, it may require more space than is allotted for the Stall.reason column and therefore can be overwritten by the Lock.ID. column output.

For more detailed lock information, perform the following steps:

- Press the L option from the horizontal menu to display a menu of lock IDs.
- Select the desired lock ID.

6.10.4 RDMS$TTB_HASH_SIZE Logical Name

The logical name RDMS$TTB_HASH_SIZE sets the size of the hash table used for temporary tables. If the logical name is not defined, Oracle Rdb uses a default value of 1249.

If you expect that temporary tables will be large (that is, 10K or more rows), use this logical name to adjust the hash table size to avoid long hash chains. Set the value to approximately 1/4 of the expected maximum number of rows for each temporary table. For example, if a temporary table will be populated with 100,000 rows, define this logical name to be 25000. If there are memory constraints on your system, you should define the logical name to be no higher than this value (1/4 of the expected maximum number of rows).

6.10.5 Error in Updating and Retrieving a Row by Dbkey Example 3−22

Example 3−22 in Section 3.8.3 that shows how to update and retrieve a row by dbkey is incorrect. The example should appear as follows:

```
SQL> ATTACH 'FILENAME MF_PERSONNEL.RDB';
SQL> --
SQL> -- Declare host variables
SQL> --
SQL> DECLARE :hv_row INTEGER;              -- Row counter
SQL> DECLARE :hv_employee_id ID_DOM;       -- EMPLOYEE_ID field
SQL> DECLARE :hv_employee_id_ind SMALLINT; -- Null indicator variable
SQL> --
SQL> DECLARE :hv_dbkey CHAR(8);            -- DBKEY storage
SQL> DECLARE :hv_dbkey_ind SMALLINT;       -- Null indicator variable
SQL> --
SQL> DECLARE :hv_last_name LAST_NAME_DOM;
SQL> DECLARE :hv_new_address_data_1 ADDRESS_DATA_1_DOM;
SQL> --
SQL> SET TRANSACTION READ WRITE;
SQL> BEGIN
cont> --
cont> -- Set the search value for SELECT
cont> --
cont>  SET :hv_last_name = 'Ames';
cont> --
cont> -- Set the NEW_ADDRESS_DATA_1 value
cont> --
cont> SET :hv_new_address_data_1 = '100 Broadway Ave.';
cont> END;
SQL> COMMIT;
SQL> --
SQL> SET TRANSACTION READ ONLY;
SQL> BEGIN
cont> SELECT E.EMPLOYEE_ID, E.DBKEY
cont>   INTO :hv_employee_id INDICATOR :hv_employee_id_ind,
cont>        :hv_dbkey INDICATOR :hv_dbkey_ind
cont>   FROM EMPLOYEES E
cont> WHERE E.LAST_NAME = :hv_last_name
cont> LIMIT TO 1 ROW;
cont> --
cont> GET DIAGNOSTICS :hv_row = ROW_COUNT;
cont> END;
SQL> COMMIT;
```

```
SQL> --
SQL> SET TRANSACTION READ WRITE RESERVING EMPLOYEES FOR SHARED WRITE;
SQL> BEGIN
cont> IF (:hv_row = 1) THEN
cont>    BEGIN
cont>    UPDATE EMPLOYEES E
cont>      SET E.ADDRESS_DATA_1 = :hv_new_address_data_1
cont>    WHERE E.DBKEY = :hv_dbkey;
cont>    END;
cont> END IF;
cont> END;
SQL> COMMIT;
SQL> --
SQL> -- Display result of change
SQL> --
SQL> SET TRANSACTION READ ONLY;
SQL> SELECT E.*
cont> FROM EMPLOYEES E
cont> WHERE E.DBKEY = :hv_dbkey;
 EMPLOYEE_ID   LAST_NAME        FIRST_NAME   MIDDLE_INITIAL
   ADDRESS_DATA_1              ADDRESS_DATA_2         CITY
      STATE   POSTAL_CODE   SEX    BIRTHDAY      STATUS_CODE
 00416         Ames             Louie        A
   100 Broadway Ave.                                 Alton
      NH      03809         M      13-Apr-1941   1

1 row selected
SQL>
```

The new example will appear in a future publication of the Oracle Rdb7 Guide to Database Performance and Tuning manual.

6.10.6 Error in Calculation of Sorted Index in Example 3−46

Example 3−46 in Section 3.9.5.1 shows the output when you use the RMU Analyze Indexes command and specify the Option=Debug qualifier and the DEPARTMENTS_INDEX sorted index.

The description of the example did not include the 8 byte dbkey in the calculation of the sorted index. The complete description is as follows:

The entire index (26 records) is located on pages 2 and 3 in logical area 72 and uses 188 bytes of a possible 430 bytes or the node record is 47 percent full. Note that due to index compression, the node size has decreased in size from 422 bytes to 188 bytes and the percent fullness of the node records has dropped from 98 to 47 percent. Also note that the used/avail value in the summary information at the end of the output does not include the index header and trailer information, which accounts for 32 bytes. This value is shown for each node record in the detailed part of the output. The number of bytes used by the index is calculated as follows: the sort key is 4 bytes plus a null byte for a total of 5 bytes. The prefix is 1 byte and the suffix is 1 byte. The prefix indicates the number of bytes in the preceding key that are the same and the suffix indicates the number of bytes that are different from the preceding key. The dbkey pointer to the row is 8 bytes. There are 26 data rows multiplied by 15 bytes for a total of 390 bytes. The 15 bytes include:

- 7 bytes for the sort key: length + null byte + prefix + suffix
- 8 bytes for the dbkey pointer to the row

Add 32 bytes for index header and trailer information for the index node to the 390 bytes for a total of 422 bytes used. Index compression reduces the number of bytes used to 188 bytes used.

The revised description will appear in a future publication of the Oracle Rdb7 Guide to Database Performance and Tuning manual.

6.10.7 Documentation Error in Section C.7

The Oracle Rdb Guide to Database Performance And Tuning, Volume 2 contains an error in Section C.7 titled Displaying Sort Statistics with the R Flag.

When describing the output from this debugging flag, bullet 9 states:

- Work File Alloc indicates how many work files were used in the sort operation. A zero (0) value indicates that the sort was accomplished completely in memory.

This is incorrect, the statistics should be described as show below:

- Work File Alloc indicates how much space (in blocks) was allocated in the work files for this sort operation. A zero (0) value indicates that the sort was accomplished completely in memory.

This error will be corrected in a future release of Oracle Rdb Guide to Database Performance And Tuning.

6.10.8 Missing Tables Descriptions for the RDBEXPERT Collection Class

Appendix B in the Oracle Rdb7 Guide to Database Performance and Tuning describes the event−based data tables in the formatted database for the Oracle Rdb PERFORMANCE and RDBEXPERT collection classes. This section describes the missing tables for the RDBEXPERT collection class.

Table 6−5 shows the TRANS_TPB table.

Table 6−5 Columns for Table EPC$1_221_TRANS_TPB

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_POINT	DATE VMS	
CLIENT_PC	INTEGER	
STREAM_ID	INTEGER	
TRANS_ID	VARCHAR(16)	
TRANS_ID_STR_ID	INTEGER	STR_ID_DOMAIN
TPB	VARCHAR(127)	
TPB_STR_ID	INTEGER	STR_ID_DOMAIN

Table 6−6 shows the TRANS_TPB_ST table. An index is provided for this table. It is defined with column STR_ID, duplicates are allowed, and the type is sorted.

Table 6−6 Columns for Table EPC$1_221_TRANS_TPB_ST

Column Name	Data Type	Domain
STR_ID	INTEGER	STR_ID_DOMAIN
SEGMENT_NUMBER	SMALLINT	SEGMENT_NUMBER_DOMAIN
STR_SEGMENT	VARCHAR(128)	

6.10.9 Missing Columns Descriptions for Tables in the Formatted Database

Some of the columns were missing from the tables in Appendix B in the Oracle Rdb7 Guide to Database Performance and Tuning. The complete table definitions are described in this section.

Table 6−7 shows the DATABASE table.

Table 6−7 Columns for Table EPC$1_221_DATABASE

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_POINT	DATE VMS	
CLIENT_PC	INTEGER	
STREAM_ID	INTEGER	
DB_NAME	VARCHAR(255)	
DB_NAME_STR_ID	INTEGER	STR_ID_DOMAIN
IMAGE_FILE_NAME	VARCHAR(255)	
IMAGE_FILE_NAME_STR_ID	INTEGER	STR_ID_DOMAIN

Table 6−8 shows the REQUEST_ACTUAL table.

Table 6−8 Columns for Table EPC$1_221_REQUEST_ACTUAL

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_START	DATE VMS	

TIMESTAMP_END	DATE VMS
DBS_READS_START	INTEGER
DBS_WRITES_START	INTEGER
RUJ_READS_START	INTEGER
RUJ_WRITES_START	INTEGER
AIJ_WRITES_START	INTEGER
ROOT_READS_START	INTEGER
ROOT_WRITES_START	INTEGER
BUFFER_READS_START	INTEGER
GET_VM_BYTES_START	INTEGER
FREE_VM_BYTES_START	INTEGER
LOCK_REQS_START	INTEGER
REQ_NOT_QUEUED_START	INTEGER
REQ_STALLS_START	INTEGER
REQ_DEADLOCKS_START	INTEGER
PROM_DEADLOCKS_START	INTEGER
LOCK_RELS_START	INTEGER
LOCK_STALL_TIME_START	INTEGER
D_FETCH_RET_START	INTEGER
D_FETCH_UPD_START	INTEGER
D_LB_ALLOK_START	INTEGER
D_LB_GBNEEDLOCK_START	INTEGER
D_LB_NEEDLOCK_START	INTEGER
D_LB_OLDVER_START	INTEGER
D_GB_NEEDLOCK_START	INTEGER
D_GB_OLDVER_START	INTEGER
D_NOTFOUND_IO_START	INTEGER
D_NOTFOUND_SYN_START	INTEGER
S_FETCH_RET_START	INTEGER
S_FETCH_UPD_START	INTEGER
S_LB_ALLOK_START	INTEGER
S_LB_GBNEEDLOCK_START	INTEGER
S_LB_NEEDLOCK_START	INTEGER
S_LB_OLDVER_START	INTEGER
S_GB_NEEDLOCK_START	INTEGER
S_GB_OLDVER_START	INTEGER
S_NOTFOUND_IO_START	INTEGER
S_NOTFOUND_SYN_START	INTEGER
D_ASYNC_FETCH_START	INTEGER
S_ASYNC_FETCH_START	INTEGER
D_ASYNC_READIO_START	INTEGER
S_ASYNC_READIO_START	INTEGER

AS_READ_STALL_START	INTEGER	
AS_BATCH_WRITE_START	INTEGER	
AS_WRITE_STALL_START	INTEGER	
BIO_START	INTEGER	
DIO_START	INTEGER	
PAGEFAULTS_START	INTEGER	
PAGEFAULT_IO_START	INTEGER	
CPU_START	INTEGER	
CURRENT_PRIO_START	SMALLINT	
VIRTUAL_SIZE_START	INTEGER	
WS_SIZE_START	INTEGER	
WS_PRIVATE_START	INTEGER	
WS_GLOBAL_START	INTEGER	
CLIENT_PC_END	INTEGER	
STREAM_ID_END	INTEGER	
REQ_ID_END	INTEGER	
COMP_STATUS_END	INTEGER	
REQUEST_OPER_END	INTEGER	
TRANS_ID_END	VARCHAR(16)	
TRANS_ID_END_STR_ID	INTEGER	STR_ID_DOMAIN
DBS_READS_END	INTEGER	
DBS_WRITES_END	INTEGER	
RUJ_READS_END	INTEGER	
RUJ_WRITES_END	INTEGER	
AIJ_WRITES_END	INTEGER	
ROOT_READS_END	INTEGER	
ROOT_WRITES_END	INTEGER	
BUFFER_READS_END	INTEGER	
GET_VM_BYTES_END	INTEGER	
FREE_VM_BYTES_END	INTEGER	
LOCK_REQS_END	INTEGER	
REQ_NOT_QUEUED_END	INTEGER	
REQ_STALLS_END	INTEGER	
REQ_DEADLOCKS_END	INTEGER	
PROM_DEADLOCKS_END	INTEGER	
LOCK_RELS_END	INTEGER	
LOCK_STALL_TIME_END	INTEGER	
D_FETCH_RET_END	INTEGER	
D_FETCH_UPD_END	INTEGER	
D_LB_ALLOK_END	INTEGER	
D_LB_GBNEEDLOCK_END	INTEGER	
D_LB_NEEDLOCK_END	INTEGER	

D_LB_OLDVER_END	INTEGER
D_GB_NEEDLOCK_END	INTEGER
D_GB_OLDVER_END	INTEGER
D_NOTFOUND_IO_END	INTEGER
D_NOTFOUND_SYN_END	INTEGER
S_FETCH_RET_END	INTEGER
S_FETCH_UPD_END	INTEGER
S_LB_ALLOK_END	INTEGER
S_LB_GBNEEDLOCK_END	INTEGER
S_LB_NEEDLOCK_END	INTEGER
S_LB_OLDVER_END	INTEGER
S_GB_NEEDLOCK_END	INTEGER
S_GB_OLDVER_END	INTEGER
S_NOTFOUND_IO_END	INTEGER
S_NOTFOUND_SYN_END	INTEGER
D_ASYNC_FETCH_END	INTEGER
S_ASYNC_FETCH_END	INTEGER
D_ASYNC_READIO_END	INTEGER
S_ASYNC_READIO_END	INTEGER
AS_READ_STALL_END	INTEGER
AS_BATCH_WRITE_END	INTEGER
AS_WRITE_STALL_END	INTEGER
BIO_END	INTEGER
DIO_END	INTEGER
PAGEFAULTS_END	INTEGER
PAGEFAULT_IO_END	INTEGER
CPU_END	INTEGER
CURRENT_PRIO_END	SMALLINT
VIRTUAL_SIZE_END	INTEGER
WS_SIZE_END	INTEGER
WS_PRIVATE_END	INTEGER
WS_GLOBAL_END	INTEGER

Table 6−9 shows the TRANSACTION table.

Table 6−9 Columns for Table EPC$1_221_TRANSACTION

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_START	DATE VMS	

TIMESTAMP_END	DATE VMS	
CLIENT_PC_START	INTEGER	
STREAM_ID_START	INTEGER	
LOCK_MODE_START	INTEGER	
TRANS_ID_START	VARCHAR(16)	
TRANS_ID_START_STR_ID	INTEGER	STR_ID_DOMAIN
GLOBAL_TID_START	VARCHAR(16)	
GLOBAL_TID_START_STR_ID	INTEGER	STR_ID_DOMAIN
DBS_READS_START	INTEGER	
DBS_WRITES_START	INTEGER	
RUJ_READS_START	INTEGER	
RUJ_WRITES_START	INTEGER	
AIJ_WRITES_START	INTEGER	
ROOT_READS_START	INTEGER	
ROOT_WRITES_START	INTEGER	
BUFFER_READS_START	INTEGER	
GET_VM_BYTES_START	INTEGER	
FREE_VM_BYTES_START	INTEGER	
LOCK_REQS_START	INTEGER	
REQ_NOT_QUEUED_START	INTEGER	
REQ_STALLS_START	INTEGER	
REQ_DEADLOCKS_START	INTEGER	
PROM_DEADLOCKS_START	INTEGER	
LOCK_RELS_START	INTEGER	
LOCK_STALL_TIME_START	INTEGER	
D_FETCH_RET_START	INTEGER	
D_FETCH_UPD_START	INTEGER	
D_LB_ALLOK_START	INTEGER	
D_LB_GBNEEDLOCK_START	INTEGER	
D_LB_NEEDLOCK_START	INTEGER	
D_LB_OLDVER_START	INTEGER	
D_GB_NEEDLOCK_START	INTEGER	
D_GB_OLDVER_START	INTEGER	
D_NOTFOUND_IO_START	INTEGER	
D_NOTFOUND_SYN_START	INTEGER	
S_FETCH_RET_START	INTEGER	
S_FETCH_UPD_START	INTEGER	
S_LB_ALLOK_START	INTEGER	
S_LB_GBNEEDLOCK_START	INTEGER	
S_LB_NEEDLOCK_START	INTEGER	
S_LB_OLDVER_START	INTEGER	
S_GB_NEEDLOCK_START	INTEGER	

S_GB_OLDVER_START	INTEGER	
S_NOTFOUND_IO_START	INTEGER	
S_NOTFOUND_SYN_START	INTEGER	
D_ASYNC_FETCH_START	INTEGER	
S_ASYNC_FETCH_START	INTEGER	
D_ASYNC_READIO_START	INTEGER	
S_ASYNC_READIO_START	INTEGER	
AS_READ_STALL_START	INTEGER	
AS_BATCH_WRITE_START	INTEGER	
AS_WRITE_STALL_START	INTEGER	
AREA_ITEMS_START	VARCHAR(128)	
AREA_ITEMS_START_STR_ID	INTEGER	STR_ID_DOMAIN
BIO_START	INTEGER	
DIO_START	INTEGER	
PAGEFAULTS_START	INTEGER	
PAGEFAULT_IO_START	INTEGER	
CPU_START	INTEGER	
CURRENT_PRIO_START	SMALLINT	
VIRTUAL_SIZE_START	INTEGER	
WS_SIZE_START	INTEGER	
WS_PRIVATE_START	INTEGER	
WS_GLOBAL_START	INTEGER	
CROSS_FAC_2_START	INTEGER	
CROSS_FAC_3_START	INTEGER	
CROSS_FAC_7_START	INTEGER	
CROSS_FAC_14_START	INTEGER	
DBS_READS_END	INTEGER	
DBS_WRITES_END	INTEGER	
RUJ_READS_END	INTEGER	
RUJ_WRITES_END	INTEGER	
AIJ_WRITES_END	INTEGER	
ROOT_READS_END	INTEGER	
ROOT_WRITES_END	INTEGER	
BUFFER_READS_END	INTEGER	
GET_VM_BYTES_END	INTEGER	
FREE_VM_BYTES_END	INTEGER	
LOCK_REQS_END	INTEGER	
REQ_NOT_QUEUED_END	INTEGER	
REQ_STALLS_END	INTEGER	
REQ_DEADLOCKS_END	INTEGER	
PROM_DEADLOCKS_END	INTEGER	
LOCK_RELS_END	INTEGER	

LOCK_STALL_TIME_END	INTEGER	
D_FETCH_RET_END	INTEGER	
D_FETCH_UPD_END	INTEGER	
D_LB_ALLOK_END	INTEGER	
D_LB_GBNEEDLOCK_END	INTEGER	
D_LB_NEEDLOCK_END	INTEGER	
D_LB_OLDVER_END	INTEGER	
D_GB_NEEDLOCK_END	INTEGER	
D_GB_OLDVER_END	INTEGER	
D_NOTFOUND_IO_END	INTEGER	
D_NOTFOUND_SYN_END	INTEGER	
S_FETCH_RET_END	INTEGER	
S_FETCH_UPD_END	INTEGER	
S_LB_ALLOK_END	INTEGER	
S_LB_GBNEEDLOCK_END	INTEGER	
S_LB_NEEDLOCK_END	INTEGER	
S_LB_OLDVER_END	INTEGER	
S_GB_NEEDLOCK_END	INTEGER	
S_GB_OLDVER_END	INTEGER	
S_NOTFOUND_IO_END	INTEGER	
S_NOTFOUND_SYN_END	INTEGER	
D_ASYNC_FETCH_END	INTEGER	
S_ASYNC_FETCH_END	INTEGER	
D_ASYNC_READIO_END	INTEGER	
S_ASYNC_READIO_END	INTEGER	
AS_READ_STALL_END	INTEGER	
AS_BATCH_WRITE_END	INTEGER	
AS_WRITE_STALL_END	INTEGER	
AREA_ITEMS_END	VARCHAR(128)	
AREA_ITEMS_END_STR_ID	INTEGER	STR_ID_DOMAIN
BIO_END	INTEGER	
DIO_END	INTEGER	
PAGEFAULTS_END	INTEGER	
PAGEFAULT_IO_END	INTEGER	
CPU_END	INTEGER	
CURRENT_PRIO_END	SMALLINT	
VIRTUAL_SIZE_END	INTEGER	
WS_SIZE_END	INTEGER	
WS_PRIVATE_END	INTEGER	
WS_GLOBAL_END	INTEGER	
CROSS_FAC_2_END	INTEGER	
CROSS_FAC_3_END	INTEGER	

CROSS_FAC_7_END	INTEGER
CROSS_FAC_14_END	INTEGER

Table 6−10 shows the REQUEST_BLR table.

Table 6−10 Columns for Table EPC$1_221_REQUEST_BLR

Column Name	Data Type	Domain
COLLECTION_RECORD_ID	SMALLINT	COLLECTION_RECORD_ID_DOMAIN
IMAGE_RECORD_ID	INTEGER	IMAGE_RECORD_ID_DOMAIN
CONTEXT_NUMBER	INTEGER	CONTEXT_NUMBER_DOMAIN
TIMESTAMP_POINT	DATE VMS	
CLIENT_PC	INTEGER	
STREAM_ID	INTEGER	
REQ_ID	INTEGER	
TRANS_ID	VARCHAR(16)	
TRANS_ID_STR_ID	INTEGER	STR_ID_DOMAIN
REQUEST_NAME	VARCHAR(31)	
REQUEST_NAME_STR_ID	INTEGER	STR_ID_DOMAIN
REQUEST_TYPE	INTEGER	
BLR	VARCHAR(127)	
BLR_STR_ID	INTEGER	STR_ID_DOMAIN

6.10.10 A Way to Find the Transaction Type of a Particular Transaction Within the Trace Database

The table EPC$1_221_TRANSACTION in the formatted Oracle Trace database has a column LOCK_MODE_START of longword datatype. The values of this column indicate the type of transaction a particular transaction was.

```
Value           Transaction type
-----           ----------------
8               Read only
9               Read write
14              Batch update
```

6.10.11 Using Oracle TRACE Collected Data

The following example shows how the OPTIMIZE AS clause is reflected in the Oracle TRACE database. When a trace collection is started the following SQL commands will record the request names.

```
SQL> attach `file personnel';
SQL> select last_name, first_name
cont> from employees
```

```
cont> optimize as request_one;
.
.
.
SQL> select employee_id
cont> from employees
cont> optimize as request_two;
.
.
.
SQL> select employee_id, city, state
cont> from employees
cont> optimize as request_three;
.
.
.
SQL> select last_name, first_name, employee_id, city, state
cont> from employees
cont> optimize as request_four;
.
.
.
```

Once an Oracle TRACE database has been populated from the collection, a query such as the following can be used to display the request names and types. The type values are described in Table 3−10. The unnamed queries in this example correspond to the queries executed by interactive SQL to validate the names of the tables an columns referenced in the user supplied queries.

```
SQL> select REQUEST_NAME, REQUEST_TYPE, TIMESTAMP_POINT
cont> from EPC$1_221_REQUEST_BLR;
REQUEST_NAME                        REQUEST_TYPE   TIMESTAMP_POINT
                                               1   15−JAN−1997 13:23:27.18
                                               1   15−JAN−1997 13:23:27.77
REQUEST_ONE                                    1   15−JAN−1997 13:23:28.21
REQUEST_TWO                                    1   15−JAN−1997 13:23:56.55
REQUEST_THREE                                  1   15−JAN−1997 13:24:57.27
REQUEST_FOUR                                   1   15−JAN−1997 13:25:25.44
6 rows selected
```

The next example shows the internal query format (BLR) converted to SQL strings after EPC$EXAMPLES:EPC_BLR_TOSQL_CONVERTER.COM has been run.

```
SQL> SELECT A.REQUEST_NAME, B.SQL_STRING FROM
cont> EPC$1_221_REQUEST_BLR A,
cont> EPC$SQL_QUERIES B
cont> WHERE A.CLIENT_PC = 0 AND A.SQL_ID = B.SQL_ID;
A.REQUEST_NAME
  B.SQL_STRING
REQUEST_ONE
      SELECT C1.LAST_NAME, C1.FIRST_NAME.        FROM EMPLOYEES C1
. . .
REQUEST_TWO
      SELECT C1.EMPLOYEE_ID.        FROM EMPLOYEES C1
. . .
REQUEST_THREE
SELECT C1.EMPLOYEE_ID, C1.CITY, C1.STATE.        FROM EMPLOYEES C1
  .
  .
  .
```

```
4 rows selected
```

Table 4−17 shows the Request Types.

Table 6−11 Request Types

Symbolic Name	Value	Comment
RDB_K_REQTYPE_OTHER	0	A query executed internally by Oracle Rdb
RDB_K_REQTYPE_USER_REQUEST	1	A non−stored SQL statement, which includes compound statements
RDB_K_REQTYPE_PROCEDURE	2	A stored procedure
RDB_K_REQTYPE_FUNCTION	3	A stored function
RDB_K_REQTYPE_TRIGGER	4	A trigger action
RDB_K_REQTYPE_CONSTRAINT	5	A table or column constraint

6.10.12 AIP Length Problems in Indexes that Allow Duplicates

When an index allows duplicates, the length stored in the AIP will be 215 bytes, regardless of the actual index node size. Because an index with duplicates can have variable node sizes, the 215−byte size is used as a median length to represent the length of rows in the index's logical area.

When the row size in the AIP is less than the actual row length, it is highly likely that SPAM entries will show space is available on pages when they have insufficient space to store another full size row. This is the most common cause of insert performance problems.

For example, consider a case where an index node size of 430 bytes (a common default value) is used; the page size for the storage area where the index is stored is 2 blocks. After deducting page overhead, the available space on a 2−block page is 982 bytes. Assume that the page in this example is initially empty.

1. A full size (430−byte) index node is stored. As 8 bytes of overhead are associated with each row stored on a page, that leaves 982−430−8 = 544 free bytes remaining on the page.
2. A duplicate key entry is made in that index node and thus a duplicate node is created on the same page. An initial duplicate node is 112 bytes long (duplicate nodes can have a variety of sizes depending on when they are created, but for this particular example, 112 bytes is used). Therefore, 544−112−8 = 424 free bytes remain on the page.

At this point, 424 bytes are left on the page. That is greater than the 215 bytes that the AIP shows as the row length for the logical area, so the SPAM page shows that the page has space available. However, an attempt to store a full size index node on the page will fail, because the remaining free space (424 bytes) is not enough to store a 430−byte node.

In this case, another candidate page must be selected via the SPAM page, and the process repeats until a page that truly has sufficient free space available is found. In a logical area that contains many duplicate nodes, a significant percentage of the pages in the logical area may fit the scenario just described. When that is the case, and a new full size index node needs to be stored, many pages may need to be read and checked before one is found that can be used to store the row.

It is possible to avoid the preceding scenario by using logical area thresholds. The goal is to set a threshold such that the SPAM page will show a page is full when space is insufficient to store a full size index node.

Using the previous example, here is how to properly set logical area thresholds to prevent excessive pages checked on an index with a 430−byte node size that is stored on a 2−block page. To calculate the proper threshold value to use, you must first determine how full the page can get before no more full size nodes will fit on the page. In this example, a database page can have up to 982−430−8 = 544 bytes in use before the page is too full. Therefore, if 544 or fewer bytes are in use, then enough space remains to store another full size node. The threshold is then 544 / 982 = .553971, or 55%.

In addition, you can determine how full a page must be before a duplicate node of size 112 will no longer fit. In this example, a database page can have up to 982−112−8 = 862 bytes in use before the page is too full. Therefore, if 862 or fewer bytes are in use, then enough space remains to store another small duplicates node. The threshold is then 862 / 982 = .8778, or 88%.

Here is an example of creating an index with the above characteristics:

```
SQL> CREATE INDEX TEST_INDEX ON EMPLOYEES (LAST_NAME)
cont>     STORE IN RDB$SYSTEM
cont>     (THRESHOLD IS (55, 55, 88));
```

These settings mean that any page at over 55% full will not be fetched when inserting a full index node, however, it may be fetched when inserting the smaller duplicates node. When the page is over 88% full then neither a full node nor a duplicate node can be stored, so the page is set as FULL. The lowest setting is not used and so can be set to any value less than or equal to the lowest used threshold.

Note that the compression algorithm used on regular tables that have compression enabled does not apply to index nodes. Index nodes are not compressed like data rows and will always utilize the number of bytes that is specified in the node size. Do not attempt to take into account a compression factor when calculating thresholds for indexes.

6.10.13 RDM$BIND_MAX_DBR_COUNT Documentation Clarification

Appendix A in Oracle Rdb7 Guide to Database Performance and Tuning incorrectly describes the use of the RDM$BIND_MAX_DBR_COUNT logical name.

Following is an updated description. Note that the difference in actual behavior between what is in the existing documentation and the software is that the logical name only controls the number of database recovery processes created at once during "node failure" recovery (that is, after a system or monitor crash or other abnormal shutdown).

When an entire database is abnormally shut down (due, for example, to a system failure), the database will have to be recovered in a "node failure" recovery mode. This recovery will be performed by another monitor in the cluster if the database is opened on another node or will be performed the next time the database is opened.

The RDM$BIND_MAX_DBR_COUNT logical name and the RDB_BIND_MAX_DBR_COUNT configuration parameter define the maximum number of database recovery (DBR) processes to be simultaneously invoked by the database monitor during a "node failure" recovery.

This logical name and configuration parameter apply only to databases that do not have global buffers enabled. Databases that utilize global buffers have only one recovery process started at a time during a "node failure" recovery.

In a node failure recovery situation with the Row Cache feature enabled (regardless of the global buffer state), the database monitor will start a single database recovery (DBR) process to recover the Row Cache Server (RCS) process and all user processes from the oldest active checkpoint in the database.

6.11 Oracle Rdb7 Guide to SQL Programming

This section provides information that is missing or changed in the Oracle Rdb7 Guide to SQL Programming.

6.11.1 Location of Host Source File Generated by the SQL Precompiler

When the SQL precompiler generates host source files (for example, .c, .pas, or .for) from the precompiler source files, it locates these files based on the Object qualifier in the command given to the SQL precompiler.

The following examples show the location where the host source file is generated.

When the Object qualifier is not specified on the command line, the object and the host source file take the name of the SQL precompiler with the extensions of .obj and .c, respectively. For example:

```
$ sqlpre/cc scc_try_mli_successful.sc
$ dir scc_try_mli_successful.*

Directory MYDISK:[LUND]

SCC_TRY_MLI_SUCCESSFUL.C;1              SCC_TRY_MLI_SUCCESSFUL.OBJ;2
SCC_TRY_MLI_SUCCESSFUL.SC;2

Total of 3 files.
```

When the Object qualifier is specified on the command line, the object and the host source take the name given on the qualifier switch. It uses the default of the SQL precompiler source if a filespec is not specified. It uses the defaults of .obj and .c if the extension is not specified. If the host language is a language other than C, it uses the appropriate host source extension (for example, .pas or .for). The files also default to the current directory if a directory specification is not specified. For example:

```
$ sqlpre/cc/obj=myobj scc_try_mli_successful.sc
$ dir scc_try_mli_successful.*

Directory MYDISK:[LUND]

SCC_TRY_MLI_SUCCESSFUL.SC;2

Total of 1 file.
$ dir myobj.*

Directory MYDISK:[LUND]

MYOBJ.C;1           MYOBJ.OBJ;2

Total of 2 files.

$ sqlpre/cc/obj=MYDISK:[lund.tmp] scc_try_mli_successful.sc
$ dir scc_try_mli_successful.*

Directory MYDISK:[LUND]

SCC_TRY_MLI_SUCCESSFUL.SC;2

Total of 1 file.
```

```
$ dir MYDISK:[lund.tmp]scc_try_mli_successful.*

Directory MYDISK:[LUND.TMP]

SCC_TRY_MLI_SUCCESSFUL.C;1              SCC_TRY_MLI_SUCCESSFUL.OBJ;2

Total of 2 files.
```

6.11.2 Remote User Authentication

In the Oracle Rdb7 Guide to SQL Programming, Table 15−1 indicates that implicit authorization works from an OpenVMS platform to another OpenVMS platform using TCP/IP. This table is incorrect. Implicit authorization only works using DECnet in this situation.

The Oracle Rdb7 Guide to SQL Programming will be fixed in a future release.

6.11.3 Additional Information About Detached Processes

Oracle Rdb documentation omits necessary detail on running Oracle Rdb from a detached process.

Applications run from detached processes must ensure that the OpenVMS environment is established correctly before running Oracle Rdb, otherwise Oracle Rdb will not execute.

Attempts to attach to a database and execute an Oracle Rdb query from applications running as detached processes will result in an error similar to the following:

```
%RDB−F−SYS_REQUEST, error from system services request
−SORT−E−OPENOUT, error opening [file] as output
−RMS−F−DEV, error in device name or inappropriate device type for operation
```

The problem occurs because a detached process does not normally have the logical names SYS$LOGIN or SYS$SCRATCH defined.

There are two methods that can be used to correct this:

- Solution 1:
  Use the DCL command procedure RUN_PROCEDURE to run the ACCOUNTS application:
  RUN_PROCEDURE.COM includes the single line:
  $ RUN ACCOUNTS_REPORT
  Then execute this procedure using this command:
  $ RUN/DETACH/AUTHORIZE SYS$SYSTEM:LOGINOUT/INPUT=RUN_PROCEDURE
  This solution executes SYS$SYSTEM:LOGINOUT so that the command language interface (DCL) is activated. This causes the logical names SYS$LOGIN and SYS$SCRATCH to be defined for the detached process. The /AUTHORIZE qualifier also ensures that the users' process quota limits (PQLs) are used from the system authorization file rather than relying on the default PQL system parameters, which are often insufficient to run Oracle Rdb.
- Solution 2:
  If DCL is not desired, and SYS$LOGIN and SYS$SCRATCH are not defined, then prior to executing any Oracle Rdb statement, you should define the following logical names:
  - RDMS$BIND_WORK_FILE

Define this logical name to allow you to reduce the overhead of disk I/O operations for matching operations when used in conjunction with the RDMS$BIND_WORK_VM logical name. If the virtual memory file is too small then overflow to disk will occur at the disk and directory location specified by RDMS$BIND_WORK_FILE.
For more information on RDMS$BIND_WORK_FILE and RDMS$BIND_WORK_VM, see the Oracle Rdb Guide to Database Performance and Tuning.

- SORTWORK0, SORTWORK1, and so on
  The OpenVMS Sort/Merge utility (SORT/MERGE) attempts to create sort work files in SYS$SCRATCH. If the SORTWORK logical names exist, the utility will not require the SYS$SCRATCH logical. However, note that not all queries will require sorting, and that some sorts will be completed in memory and so will not necessarily require disk space.
  If you use the logical RDMS$BIND_SORT_WORKFILES, you will need to define further SORTWORK logical names as described in the Oracle Rdb Guide to Database Performance and Tuning.
  You should also verify that sufficient process quotas are specified on the RUN/DETACH command line, or defined as system PQL parameters to allow Oracle Rdb to execute.

6.12 Guide to Using Oracle SQL/Services Client APIs

The following information describes Oracle SQL/Services documentation errors or omissions.

- The Guide to Using Oracle SQL/Services Client APIs does not describe changes to size and format of integer and floating−point data types
  Beginning with Oracle SQL/Services V5.1, the size and format of some integer and floating−point data types is changed as follows:
  - Trailing zeros occur in fixed−point numeric data types with SCALE FACTOR.
    Trailing zeros are now included after the decimal point up to the number of digits specified by the SCALE FACTOR. In versions of Oracle SQL/Services previous to V5.1, at most one trailing zero was included where the value was a whole number.
    The following examples illustrate the changes using a field defined as INTEGER(3):

```
V5.1 and     Versions previous
higher       to V5.1
--------     -----------------
  1.000        1.0
 23.400       23.4
567.890      567.89
```

  - Trailing zeros occur in floating−point data types. Trailing zeros are now included in the fraction, and leading zeros are included in the exponent, up to the maximum precision available, for fields assigned the REAL and DOUBLE PRECISION data types.

```
                                                Versions previous
Data Type           V5.1 and higher             to V5.1
---------------     ----------------------      -----------------
REAL                1.2340000E+01               1.234E+1
DOUBLE PRECISION    5.678900000000000E+001      5.6789E+1
```

  - Size of TINYINT and REAL data types is changed.
    The maximum size of the TINYINT and REAL data types is changed to correctly reflect the precision of the respective data types.
    The following table shows the maximum lengths of the data types now and in previous versions:

```
                    V5.1 and    Versions previous
Data type           higher      to V5.1
----------          --------    -----------------
TINYINT              4           6
REAL                15          24
```

- The Guide to Using Oracle SQL/Services Client APIs does not describe that the sqlsrv_associate() service returns SQL error code −1028 when connecting to a database service if the user has not been granted the right to attach to the database.
  When a user connects to a database service, the sqlsrv_associate() service completes with the SQL error code −1028, SQL_NO_PRIV, if the user has been granted access to the Oracle SQL/Services service, but has not been granted the right to attach to the database. A record of the failure is written to the executor process's log file. Note that the sqlsrv_associate() service completes with the Oracle SQL/Services error code −2034, SQLSRV_GETACCINF if the user has not been granted access to the Oracle SQL/Services service.

6.13 Updates to System Relations

The following sections include updates to system relations that were inadvertently omitted in the SQL Help and Rdb Help files in Release 7.0.

6.13.1 Clarification on Updates to the RDB$LAST_ALTERED Column for the RDB$DATABASE System Relation

The ALTER DATABASE statement can be used to change many database attributes, however, only those listed below will cause the RDB$DATABASE system relation to be changed. The column RDB$LAST_UPDATED is used to record the date and time when the system relation RDB$DATABASE is updated and so will change when any of the following clauses are used by ALTER DATABASE.

- CARDINALITY COLLECTION IS { ENABLED | DISABLED }
- DICTIONARY IS [NOT] REQUIRED
- DICTIONARY IS NOT USED
- METADATA CHANGES ARE { ENABLED | DISABLED }
- MULTISCHEMA IS { ON | OFF }
- SECURITY CHECKING IS EXTERNAL (PERSONAL SUPPORT IS { ENABLED | DISABLED })
- SYNONYMS ARE ENABLED
- WORKLOAD COLLECTION IS { ENABLED | DISABLED }

In addition any GRANT and REVOKE statements which use the ON DATABASE clause will cause the RDB$LAST_UPDATED column to be updated for RDB$DATABASE.

6.13.2 Missing Descriptions of RDB$FLAGS

The HELP file for Oracle Rdb describes the system relations for Oracle Rdb and was missing these updated descriptions of the RDB$FLAGS column for several system relations.

Table 6−12 Changed Columns for RDB$INDICES Table

Column Name	Data Type	Domain Name	Comments
RDB$FLAGS	integer	RDB$FLAGS	A bit mask where the bits have the following meaning when set: • Bit 0: This index is of type HASHED. • Bit 1: This index uses the MAPPING VALUES clause to compress integer value ranges. • Bit 2: If this is a HASHED index then it is of type ORDERED. If clear this indicates the index if of type SCATTERED. • Bit 3: Reserved for future use. • Bit 4: This index has run length compression enabled (ENABLE

			COMPRESSION). • Bit 5: This index is no longer used (MAINTENANCE IS DISABLED). • Bit 6 through 10: Reserved for future use. • Bit 11: This index has duplicates compressed (DUPLICATES ARE COMPRESSED). • Bit 12: This index is of type SORTED RANKED. • Bits 13 through 31: Reserved for future use.

Table 6−13 Changed Columns for RDB$RELATIONS Table

Column Name	Data Type	Domain Name	Comments
RDB$FLAGS	integer	RDB$FLAGS	A bit mask where the bits have the following meaning when set: • Bit 0: This relation is a view. • Bit 1: This relation is ***not*** compressed. • Bit 2: The SQL clause, WITH CHECK OPTION, is used in this view definition. • Bit 3: Indicates a special internal system relation. • Bit 4: This view is not an ANSI updatable view. • Bit 5: This is an imported table in the Distributed Option for Rdb catalog. • Bit 6: This is a passthru table in the Distributed Option for Rdb catalog. • Bit 7: This is a partitioned view in the Distributed Option for Rdb catalog. • Bit 8: This table has compression defined by the storage map. When set Bit 1 in this bit mask is ignored. • Bit 9: This is a temporary table. • Bit 10: When bit 9 is set this is a global temporary table, when clear it indicates a local temporary table. • Bit 11: When bit 9 is set this indicates that the rows in the temporary table should be deleted upon COMMIT. • Bit 12: Reserved for future use. • Bit 13: A table (via a computed by column) or view references a local temporary table. • Bit 14: Reserved for future use. • Bit 15: This is a system table with a special storage map. • Bits 16 through 31: Reserved for future use.

Table 6−14 Changed Columns for RDB$STORAGE_MAPS Table

Column Name	Data Type	Domain Name	Comments
RDB$FLAGS	integer	RDB$FLAGS	A bit mask where the bits have the following meaning when set: • Bit 0: This table or index is mapped to page format MIXED areas. • Bit 1: This partition is ***not*** compressed. • Bit 2: This is a strictly partitioned storage map, the partitioning columns become read only for UPDATE. • Bit 3 through 31: Reserved for future use.

6.14 Error Messages

The following subsections further describe or clarify error messages.

6.14.1 Clarification of the DDLDONOTMIX Error Message

The ALTER DATABASE statement performs two classes of functions: changing the database root structures in the .RDB file and modifying the system metadata in the RDB$SYSTEM storage area. The first class of changes do not require a transaction to be active. However, the second class requires that a transaction be active. Oracle Rdb does not currently support the mixing of these two classes of ALTER DATABASE clauses.

When you mix clauses that fall into both classes, the error message DDLDONOTMIX "the {SQL−syntax} clause can not be used with some ALTER DATABASE clauses" is displayed, and the ALTER DATABASE statement fails.

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

The following clauses may be mixed with each other but may not appear with other clauses such as ADD STORAGE AREA or ADD CACHE:

- DICTIONARY IS [NOT] REQUIRED
- DICTIONARY IS NOT USED
- MULTISCHEMA IS { ON | OFF }
- CARDINALITY COLLECTION IS { ENABLED | DISABLED }
- METADATA CHANGES ARE { ENABLED | DISABLED }
- WORKLOAD COLLECTION IS { ENABLED | DISABLED }

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

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

Chapter 7
Known Problems and Restrictions

This chapter describes problems and restrictions relating to Oracle Rdb Release 7.1.0.3, and includes workarounds where appropriate.

7.1 Known Problems and Restrictions in All Interfaces

This section describes known problems and restrictions that affect all interfaces for Release 7.1.0.3.

7.1.1 RMU/BACKUP MAX_FILE_SIZE Option Has Been Disabled

The MAX_FILE_SIZE option of the RMU/BACKUP/DISK_FILE qualifier for backup to multiple disk files has been temporarily disabled since it creates corrupt RBF files if the maximum file size in megabytes is exceeded and a new RBF file is created. It also does not give a unique name to the new RBF file but creates an RBF file with the same name but a new version number in the same disk directory. This will cause an RMU−F−BACFILCOR error on the restore and the restore will not complete.

The multi−file disk backup and restore will succeed if this option is not used. If this option is specified, a warning message is now output that this qualifier will be ignored.

The following example shows that the MAX_FILE_SIZE option, when used with the /DISK_FILE qualifier on an RMU/BACKUP, will be ignored and a warning message will be output.

```
$ RMU/BACKUP /ONLINE                    −
             /NOCRC                     −
             /NOLOG                     −
             /NOINCREMENTAL             −
             /QUIET_POINT               −
             TEST_DB_DIR:TEST_DB
  −
BACKUP_DIR_1:TEST_DB/DISK_FILE=(WRITER_THREADS=3,MAX_FILE_SIZE=10) ,−
BACKUP_DIR_2:/DISK_FILE=(WRITER_THREADS=3,MAX_FILE_SIZE=10) ,−
BACKUP_DIR_3:/DISK_FILE=(WRITER_THREADS=3,MAX_FILE_SIZE=10)

%RMU−W−DISABLEDOPTION, The MAX_FILE_SIZE option is temporarily disabled
        and will be ignored
```

As a workaround to avoid this problem, do not specify the MAX_FILE_SIZE option with the /DISK_FILE qualifier.

7.1.2 Oracle Rdb and OpenVMS ODS−5 Volumes

The OpenVMS Version 7.2 release introduced an Extended File Specifications feature, which consists of two major components:

- A new, optional, volume structure, ODS−5, which provides support for file names that are longer and have a greater range of legal characters than in previous versions of OpenVMS.
- Support for "deep" directory trees.

ODS−5 was introduced primarily to provide enhanced file sharing capabilities for users of Advanced Server for OpenVMS 7.2 (formerly known as PATHWORKS for OpenVMS), as well as DCOM and JAVA applications.

In some cases, Oracle Rdb performs its own file and directory name parsing and explicitly requires ODS−2 (the traditional OpenVMS volume structure) file and directory name conventions to be followed. Because of

this knowledge, Oracle does not support any Oracle Rdb database file components (including root files, storage area files, after image journal files, record cache backing store files, database backup files, after image journal backup files, etc.) that utilize any non−ODS−2 file naming features. For this reason, Oracle recommends that Oracle Rdb database components not be located on ODS−5 volumes.

Oracle does support Oracle Rdb database file components on ODS−5 volumes provided that all of these files and directories used by Oracle Rdb strictly follow the ODS−2 file and directory name conventions. In particular, all file names must be specified entirely in uppercase and "special" characters in file or directory names are forbidden.

7.1.3 Optimization of Check Constraints

Bug 1448422

When phrasing constraints using the "CHECK" syntax, a poorer strategy can be chosen by the optimizer than when the same or similar constraint is phrased using referential integrity (PRIMARY and FOREIGN KEY) constraints.

For example, I have two tables T1 and T2, both with one column, and I wish to ensure that all values in table T1 exist in T2. Both tables have an index on the referenced field. I could use a PRIMARY KEY constraint on T2 and a FOREIGN KEY constraint on T1.

```
SQL> alter table t2
cont>   alter column f2 primary key not deferrable;
SQL> alter table t1
cont>   alter column f1 references t2 not deferrable;
```

When deleting from the PRIMARY KEY table, Rdb will only check for rows in the FOREIGN KEY table where the FOREIGN KEY has the deleted value. This can be seen as an index lookup on T1 in the retrieval strategy.

```
SQL> delete from t2 where f2=1;
Get     Temporary relation      Retrieval by index of relation T2
  Index name  I2 [1:1]
Index only retrieval of relation T1
  Index name  I1 [1:1]
%RDB−E−INTEG_FAIL, violation of constraint T1_FOREIGN1 caused operation to fail
```

The failure of the constraint is not important. What is important is that Rdb efficiently detects that only those rows in T1 with the same values as the deleted row in T2 can be affected.

It is necessary sometimes to define this type of relationship using CHECK constraints. This could be necessary because the presence of NULL values in the table T2 precludes the definition of a primary key on that table. This could be done with a CHECK constraint of the form:

```
SQL> alter table t1
cont>   alter column f1
cont>   check (f1 in (select * from t2)) not deferrable;
SQL> delete from t2 where f2=1;
Get     Temporary relation      Retrieval by index of relation T2
  Index name  I2 [1:1]
Cross block of 2 entries
  Cross block entry 1
    Index only retrieval of relation T1
```

```
      Index name  I1 [0:0]
  Cross block entry 2
    Conjunct        Aggregate-F1    Conjunct
    Index only retrieval of relation T2
      Index name  I2 [0:0]
%RDB-E-INTEG_FAIL, violation of constraint T1_CHECK1 caused operation to fail
```

The cross block is for the constraint evaluation. This retrieval strategy indicates that to evaluate the constraint, the entire index on table T1 is being scanned and for each key, the entire index in table T2 is being scanned. The behavior can be improved somewhat by using an equality join condition in the select clause of the constraint:

```
SQL> alter table t1
cont>   alter column f1
cont>   check (f1 in (select * from t2 where f2=f1))
cont>      not deferrable;

or:

SQL> alter table t1
cont>   alter column f1
cont>   check (f1=(select * from t2 where f2=f1))
cont>      not deferrable;
```

In both cases the retrieval strategy will look like this:

```
SQL> delete from t2 where f2=1;
Get     Temporary relation      Retrieval by index of relation T2
  Index name  I2 [1:1]
Cross block of 2 entries
  Cross block entry 1
    Index only retrieval of relation T1
      Index name  I1 [0:0]
  Cross block entry 2
    Conjunct        Aggregate-F1    Conjunct
    Index only retrieval of relation T2
      Index name  I2 [1:1]
%RDB-E-INTEG_FAIL, violation of constraint T1_CHECK1 caused operation to fail
```

While the entire T1 index is scanned, at least the value from T1 is used to perform an index lookup on T2.

These restrictions result from semantic differences in the behavior of the "IN" and "EXISTS" operators with respect to null handling, and the complexity of dealing with non-equality join conditions.

To improve the performance of this type of integrity check on larger tables, it is possible to use a series of triggers to perform the constraint check. The following triggers perform a similar check to the constraints above.

```
SQL> create trigger t1_insert
cont>  after insert on t1
cont>  when (not exists (select * from t2 where f2=f1))
cont>    (error) for each row;
SQL> create trigger t1_update
cont>  after update on t1
cont>  when (not exists (select * from t2 where f2=f1))
cont>    (error) for each row;
SQL> ! A delete trigger is not needed on T1.
```

```
SQL> create trigger t2_delete
cont>  before delete on t2
cont>  when (exists (select * from t1 where f1=f2))
cont>    (error) for each row;
SQL> create trigger t2_modify
cont>  after update on t2
cont>  referencing old as t2o new as t2n
cont>  when (exists (select * from t1 where f1=t2o.f2))
cont>    (error) for each row;
SQL> ! An insert trigger is not needed on T2.
```

The strategy for a delete on T2 is now:

```
SQL> delete from t2 where f2=1;
Aggregate-F1    Index only retrieval of relation T1
  Index name  I1 [1:1]
Temporary relation      Get     Retrieval by index of relation T2
  Index name  I2 [1:1]
%RDB-E-TRIG_INV_UPD, invalid update; encountered error condition defined for
trigger
-RDMS-E-TRIG_ERROR, trigger T2_DELETE forced an error
```

The trigger strategy is the index only retrieval displayed first. You will note that the index on T1 is used to examine only those rows that may be affected by the delete.

Care must be taken when using this workaround as there are semantic differences in the operation of the triggers, the use of "IN" and "EXISTS", and the use of referential integrity constraints.

This workaround is useful where the form of the constraint is more complex, and cannot be phrased using referential integrity constraints. For example, if the application is such that the value in table T1 may be spaces or NULL to indicate the absence of a value, the above triggers could easily be modified to allow for these semantics.

7.1.4 Using Databases from Releases Earlier Than V6.0

You cannot convert or restore databases earlier than V6.0 directly to V7.1. The RMU Convert command for V7.1 supports conversions from V6.0 through V7.0 only. If you have a V3.0 through V5.1 database, you must convert it to at least V6.0 and then convert it to V7.1. For example, if you have a V4.2 database, convert it first to at least V6.0, then convert the resulting database to V7.1.

If you attempt to convert a database created prior to V6.0 directly to V7.1, Oracle RMU generates an error.

7.1.5 PAGE TRANSFER VIA MEMORY Disabled

Oracle internal testing has revealed that the PAGE TRANSFER VIA MEMORY option for global buffers is not as robust as is needed for the mission critical environments where Oracle Rdb7 is often deployed. This feature has been disabled in release 7.1. Oracle intends to re−enable this feature in a future release.

7.1.6 Carryover Locks and NOWAIT Transaction Clarification

In NOWAIT transactions, the BLAST (Blocking AST) mechanism cannot be used. For the blocking user to receive the BLAST signal, the requesting user must request the locked resource with WAIT (which a

NOWAIT transaction does not do). Oracle Rdb defines a resource called NOWAIT, which is used to indicate that a NOWAIT transaction has been started. When a NOWAIT transaction starts, the user requests the NOWAIT resource. All other database users hold a lock on the NOWAIT resource so that when the NOWAIT transaction starts, all other users are notified with a NOWAIT BLAST. The BLAST causes blocking users to release any carryover locks. There can be a delay before the transactions with carryover locks detect the presence of the NOWAIT transaction and release their carryover locks. You can detect this condition by examining the stall messages. If the "Waiting for NOWAIT signal (CW)" stall message appears frequently, the application is probably experiencing a decrease in performance, and you should consider disabling the carryover lock behavior.

7.1.7 Unexpected Results Occur During Read−Only Transactions on a Hot Standby Database

When using Hot Standby, it is typical to use the standby database for reporting, simple queries, and other read−only transactions. If you are performing these types of read−only transactions on a standby database, be sure you can tolerate a READ COMMIT level of isolation. This is because the Hot Standby database might be updated by another transaction before the read−only transaction finishes, and the data retrieved might not be what you expected.

Because Hot Standby does not write to the snapshot files, the isolation level achieved on the standby database for any read−only transaction is a READ COMMITED transaction. This means that nonrepeatable reads and phantom reads are allowed during the read−only transaction:

- Nonrepeatable read operations: Allows the return of different results within a single transaction when an SQL operation reads the same row in a table twice. Nonrepeatable reads can occur when another transaction modifies and commits a change to the row between transactions. Because the standby database will update the data when it confirms a transaction has been committed, it is very possible to see an SQL operation on a standby database return different results.
- Phantom read operations: Allows the return of different results within a single transaction when an SQL operation retrieves a range of data values (or similar data existence check) twice. Phantoms can occur if another transaction inserted a new record and committed the insertion between executions of the range retrieval. Again, because the standby database may do this, phantom reads are possible.

Thus, you cannot rely on any data read from the standby database to remain unchanged. Be sure your read−only transactions can tolerate a READ COMMIT level of isolation before you implement procedures that read and use data from a standby database.

7.1.8 IMPORT Unable to Import Some View Definitions

View definitions that reference SQL functions, created by the CREATE MODULE statement, cannot be imported by the SQL IMPORT statement. This is because the views are defined before the functions themselves exist.

The following example shows the errors from IMPORT:

```
IMPORTing view TVIEW
%SQL−F−NOVIERES, unable to import view TVIEW
%RDB−E−NO_META_UPDATE, metadata update failed
−RDB−E−OBSOLETE_METADA, request references metadata objects that no
longer exist
−RDMS−E−RTNNEXTS, routine FORMAT_OUT does not exist in this database
```

```
%RDB-E-OBSOLETE_METADA, request references metadata objects that no
longer exist
-RDMS-F-TABNOTDEF, relation TVIEW is not defined in database
```

The following script can be used to demonstrate the problem:

```
create database filename badimp;

create table t (sex char);

create module TFORMAT
    language SQL

    function FORMAT_OUT (:s char)
    returns char(4);
    return (case :s
    when 'F' then 'Female'
    when 'M' then 'Male'
    else NULL
    end);
end module;

create view TVIEW (m_f) as
    select FORMAT_OUT (sex) from t;

commit;

export database filename badimp into exp;
drop database filename badimp;
import database from exp filename badimp;
```

This restriction will be lifted in a future release of Oracle Rdb. Currently the workaround is to save the view definitions and reapply them after the import operation completes.

This restriction does not apply to external functions, created using the CREATE FUNCTION statement, as these database objects are defined before tables and views.

7.1.9 Both Application and Oracle Rdb Using SYS$HIBER

In application processes that use Oracle Rdb and the $HIBER system service (possibly through RTL routines such as LIB$WAIT), the application must ensure that the event being waited for has actually occurred. Oracle Rdb uses $HIBER/$WAKE sequences for interprocess communications particularly when the ALS (AIJ Log Server) feature is enabled.

The use of the $WAKE system service by Oracle Rdb can interfere with other users of $HIBER (such as the routine LIB$WAIT) that do not check for event completion, possibly causing a $HIBER to be unexpectedly resumed without waiting at all.

To avoid these situations, consider altering the application to use a code sequence that avoids continuing without a check for the operation (such as a delay or a timer firing) being complete.

The following pseudo−code shows how a flag can be used to indicate that a timed−wait has completed correctly. The wait does not complete until the timer has actually fired and set TIMER_FLAG to TRUE. This code relies on ASTs being enabled.

```
ROUTINE TIMER_WAIT:
    BEGIN
    ! Clear the timer flag
    TIMER_FLAG = FALSE
    ! Schedule an AST for sometime in the future
    STAT = SYS$SETIMR (TIMADR = DELTATIME, ASTRTN = TIMER_AST)
    IF STAT <> SS$_NORMAL
    THEN BEGIN
         LIB$SIGNAL (STAT)
         END
    ! Hibernate.  When the $HIBER completes, check to make
    ! sure that TIMER_FLAG is set indicating that the wait
    ! has finished.
    WHILE TIMER_FLAG = FALSE
    DO  BEGIN
        SYS$HIBER()
        END
    END
ROUTINE TIMER_AST:
    BEGIN
    ! Set the flag indicating that the timer has expired
    TIMER_FLAG = TRUE
    ! Wake the main-line code
    STAT = SYS$WAKE ()
    IF STAT <> SS$_NORMAL
    THEN BEGIN
         LIB$SIGNAL (STAT)
         END
    END
```

The LIB$K_NOWAKE flag can be specified when using the OpenVMS LIB$WAIT routine to allow an alternate wait scheme (using the $SYNCH system service) that can avoid potential problems with multiple code sequences using the $HIBER system service.

7.1.10 Bugcheck Dump Files with Exceptions at COSI_CHF_SIGNAL

In certain situations, Oracle Rdb bugcheck dump files indicate an exception at COSI_CHF_SIGNAL. This location is, however, not the address of the actual exception. The actual exception occurred at the previous call frame on the stack (the one listed as the next Saved PC after the exception).

For example, consider the following bugcheck file stack information:

```
$ SEARCH RDSBUGCHK.DMP "EXCEPTION","SAVED PC","-F-","-E-"

***** Exception at 00EFA828 : COSI_CHF_SIGNAL + 00000140
%COSI-F-BUGCHECK, internal consistency failure
Saved PC = 00C386F0 : PSIINDEX2JOINSCR + 00000318
Saved PC = 00C0BE6C : PSII2BALANCE + 0000105C
Saved PC = 00C0F4D4 : PSII2INSERTT + 000005CC
Saved PC = 00C10640 : PSII2INSERTTREE + 000001A0
      .
      .
      .
```

In this example, the exception actually occurred at PSIINDEX2JOINSCR offset 00000318. If you have a

bugcheck dump with an exception at COSI_CHF_SIGNAL, it is important to note the next "Saved PC" because it is needed when working with Oracle Rdb Worldwide Support.

7.1.11 Read−only Transactions Fetch AIP Pages Too Often

Oracle Rdb read−only transactions fetch Area Inventory Pages (AIP) to ensure that the logical area has not been modified by an exclusive read−write transaction. This check is needed because an exclusive read−write transaction does not write snapshot pages and these pages may be needed by the read−only transaction.

Because AIPs are always stored in the RDB$SYSTEM area, reading the AIP pages could represent a significant amount of I/O to the RDB$SYSTEM area for some applications. Setting the RDB$SYSTEM area to read−only can avoid this problem, but it also prevents other online operations that might be required by the application so it is not a viable workaround in all cases.

This problem has been reduced in Oracle Rdb release 7.0. The AIP entries are now read once and then are not read again unless they need to be. This optimization requires that the carry−over locks feature be enabled (this is the default setting). If carry over locks are not enabled, this optimization is not enabled and the behavior is the same as in previous releases.

7.1.12 Row Cache Not Allowed While Hot Standby Replication is Active

The row cache feature may not be enabled on a hot standby database while replication is active. The hot standby feature will not start if row cache is enabled.

This restriction exists because rows in the row cache are accessed via logical dbkeys. However, information transferred to the hotstandby database via the after image journal facility only contains physical dbkeys. Because there is no way to maintain rows in the cache via the hot standby processing, the row cache must be disabled when the standby database is open and replication is active.

A new command qualifier, ROW_CACHE=DISABLED, has been added to the RMU Open command. To open the hot standby database prior to starting replication, use the ROW_CACHE=DISABLED qualifier on the RMU Open command.

7.1.13 Excessive Process Page Faults and other Performance Considerations During Oracle Rdb Sorts

Excessive hard or soft page faulting can be a limiting factor of process performance. One factor contributing to Oracle Rdb process page faulting is sorting operations. Common causes of sorts include the SQL GROUP BY, ORDER BY, UNION, and DISTINCT clauses specified for a query, and index creation operations. Defining the logical name RDMS$DEBUG_FLAGS to "RS" can help determine when Oracle Rdb sort operations are occurring and to display the sort keys and statistics.

Oracle Rdb includes its own copy of the OpenVMS SORT32 code within the Oracle Rdb images and does not generally call the routines in the OpenVMS run−time library. A copy of the SORT32 code is used to provide stability between versions of Oracle Rdb and OpenVMS and because Oracle Rdb calls the sort routines from executive processor mode which is difficult to do using the SORT32 shareable image. SQL IMPORT and RMU Load operations do, however, call the OpenVMS SORT run−time library.

At the beginning of a sort operation, the SORT code allocates some memory for working space. The SORT code uses this space for buffers, in−memory copies of the data, and sorting trees.

SORT does not directly consider the processes quotas or parameters when allocating memory. The effects of WSQUOTA and WSEXTENT are indirect. At the beginning of each sort operation, the SORT code attempts to adjust the process working set to the maximum possible size using the $ADJWSL system service specifying a requested working set limit of %X7FFFFFFF pages (the maximum possible). SORT then uses a value of 75% of the returned working set for virtual memory scratch space. The scratch space is then initialized and the sort begins.

The initialization of the scratch space generally causes page faults to access the pages newly added to the working set. Pages that were in the working set already may be faulted out as the new pages are faulted in. Once the sort operation completes and SORT returns back to Oracle Rdb, the pages that may have been faulted out of the working set are likely to be faulted back into the working set.

When a process working set is limited by the working set quota (WSQUOTA) parameter and the working set extent (WSEXTENT) parameter is a much larger value, the first call to the sort routines can cause many page faults as the working set grows. Using a value of WSEXTENT that is closer to WSQUOTA can help reduce the impact of this case.

With some OpenVMS versions, AUTOGEN sets the SYSGEN parameter PQL_MWSEXTENT equal to the WSMAX parameter. This means that all processes on the system end up with WSEXTENT the same as WSMAX. Since that might be quite high, sorting might result in excessive page faulting. You may want to explicitly set PQL_MWSEXTENT to a lower value if this is the case on your system.

Sort work files are another factor to consider when tuning for Oracle Rdb sort operations. When the operation can not be done in the available memory, SORT uses temporary disk files to hold the data as it is being sorted. The Oracle Rdb7 Guide to Database Performance and Tuning contains more detailed information about sort work files.

The logical name RDMS$BIND_SORT_WORKFILES specifies how many work files sort is to use if work files are required. The default is 2 and the maximum number is 10. The work files can be individually controlled by the SORTWORKn logical names (where n is from 0 through 9). You can increase the efficiency of sort operations by assigning the location of the temporary sort work files to different disks. These assignments are made by using up to ten logical names, SORTWORK0 through SORTWORK9.

Normally, SORT places work files in the your SYS$SCRATCH directory. By default, SYS$SCRATCH is the same device and directory as the SYS$LOGIN location. Spreading the I/O load over many disks improves efficiency as well as performance by taking advantage of the system resources and helps prevent disk I/O bottlenecks. Specifying that a your work files reside on separate disks permits overlap of the SORT read/write cycle. You may also encounter cases where insufficient space exists on the SYS$SCRATCH disk device (for example, while Oracle Rdb builds indexes for a very large table). Using the SORTWORK0 through SORTWORK9 logical names can help you avoid this problem.

Note that SORT uses the work files for different sorted runs, and then merges the sorted runs into larger groups. If the source data is mostly sorted, then not every sort work file may need to be accessed. This is a possible source of confusion because even with 10 sort work files, it is possible to exceed the capacity of the first SORT file and the sort operation fails never having accessed the remaining 9 sort work files.

Note that the logical names RDMS$BIND_WORK_VM and RDMS$BIND_WORK_FILE do not affect or control the operation of sort. These logical names are used to control other temporary space allocation within

Oracle Rdb.

7.1.14 Control of Sort Work Memory Allocation

Oracle Rdb uses a built−in SORT32 package to perform many sort operations. Sometimes, these sorts exhibit a significant performance problem when initializing work memory to be used for the sort. This behavior can be experienced, for example, when a very large sort cardinality is estimated, but the actual sort cardinality is small.

In rare cases, it may be desirable to artificially limit the sort package's use of work memory. Two logicals have been created to allow this control. In general, there should be no need to use either of these logicals and misuse of them can significantly impact sort performance. Oracle recommends that these logicals be used carefully and sparingly.

The logical names are:

Table 7−1 Sort Memory Logicals

Logical	Definition
RDMS$BIND_SORT_MEMORY_WS_FACTOR	Specifies a percentage of the process's working set limit to be used when allocating sort memory for the built−in SORT32 package. If not defined, the default value is 75 (representing 75%), the maximum value is 75 (representing 75%), and the minimum value is 2 (representing 2%). Processes with vary large working set limits can sometimes experience significant page faulting and CPU consumption while initializing sort memory. This logical name can restrict the sort work memory to a percentage of the processes maximum working set.
RDMS$BIND_SORT_MEMORY_MAX_BYTES	Specifies an absolute limit to be used when allocating sort memory for the built−in SORT32 package. If not defined, the default value is unlimited (up to 1GB), the maximum value is 2,147,483,647 and the minimum value is 32,768.

7.1.15 The Halloween Problem

When a cursor is processing rows selected from a table, it is possible that another separate query can interfere with the retrieval of the cursor by modifying the index columns key values used by the cursor.

For instance, if a cursor selects all EMPLOYEES with LAST_NAME >= 'M', it is likely that the query will use the sorted index on LAST_NAME to retrieve the rows for the cursor. If an update occurs during the processing of the cursor which changes the LAST_NAME of an employee from "Mason" to "Rickard", then it is possible that that employee row will be processed twice. First when it is fetched with name "Mason", and then later when it is accessed by the new name "Rickard".

The Halloween problem is a well known problem in relational databases. Access strategies which optimize the I/O requirements, such as Index Retrieval, can be subject to this problem. Interference from queries by other sessions are avoided by locking and are controlled by the ISOLATION LEVEL options in SQL, or the CONCURRENCY/CONSISTENCY options in RDO/RDML.

Oracle Rdb avoids this problem if it knows that the cursors subject table will be updated. For example, if the SQL syntax UPDATE ... WHERE CURRENT OF is used to perform updates of target rows, or the RDO/RDML MODIFY statement uses the context variable for the stream. Then the optimizer will choose an alternate access strategy if an update can occur which may cause the Halloween problem. This can be seen in the access strategy in Example 2−2 as a "Temporary relation" being created to hold the result of the cursor query.

When you use interactive or dynamic SQL, the UPDATE ... WHERE CURRENT OF or DELETE ... WHERE CURRENT OF statements will not be seen until after the cursor is declared and opened. In these environments, you must use the FOR UPDATE clause to specify that columns selected by the cursor will be updated during cursor processing. This is an indication to the Rdb optimizer so that it protects against the Halloween problem in this case. This is shown in Example 2−1 and Example 2−2.

The following example shows that the EMP_LAST_NAME index is used for retrieval. Any update performed will possibly be subject to the Halloween problem.

```
SQL> set flags 'strategy';
SQL> declare emp  cursor for
cont> select * from employees where last_name >= 'M'
cont> order by last_name;
SQL> open emp;
Conjunct        Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_LAST_NAME [1:0]
SQL> close emp;
```

The following example shows that the query specifies that the column LAST_NAME will be updated by some later query. Now the optimizer protects the EMP_LAST_NAME index used for retrieval by using a "Temporary Relation" to hold the query result set. Any update performed on LAST_NAME will now avoid the Halloween problem.

```
SQL> set flags 'strategy';
SQL> declare emp2 cursor for
cont> select * from employees where last_name >= 'M'
cont> order by last_name
cont> for update of last_name;
SQL> open emp2;
Temporary relation      Conjunct        Get
Retrieval by index of relation EMPLOYEES
  Index name  EMP_LAST_NAME [1:0]
SQL> close emp2;
```

When you use the SQL precompiler, or the SQL module language compiler it can be determined from usage that the cursor context will possibly be updated during the processing of the cursor because all cursor related statements are present within the module. This is also true for the RDML/RDBPRE precompilers when you use the DECLARE_STREAM and START_STREAM statements and use the same stream context to perform all MODIFY and ERASE statements.

The point to note here is that the protection takes place during the open of the SQL cursor (or RDO stream), not during the subsequent UPDATE or DELETE.

If you execute a separate UPDATE query which modifies rows being fetched from the cursor then the actual rows fetched will depend upon the access strategy chosen by the Rdb optimizer. As the query is separate from the cursors query (i.e. doesn't reference the cursor context), then the optimizer does not know that the cursor selected rows are potentially updated and so cannot perform the normal protection against the Halloween problem.

7.2 SQL Known Problems and Restrictions

This section describes known problems and restrictions for the SQL interface for release 7.1.

7.2.1 Interchange File (RBR) Created by Oracle Rdb Release 7.1 Not Compatible With Previous Releases

To support the large number of new database attributes and objects, the protocol used by SQL EXPORT and SQL IMPORT has been enhanced to support more protocol types. Therefore, this format of the Oracle Rdb release 7.1 interchange files can no longer be read by older versions of Oracle Rdb.

Oracle Rdb continues to provide upward compatibility for interchange files generated by older versions.

Oracle Rdb has never supported backward compatibility, however, it was sometimes possible to use an interchange file with an older version of IMPORT. However, this protocol change will no longer permit this usage.

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

The SQL DROP MODULE ... CASCADE statement may sometimes generate an unexpected NO_META_UPDATE error. This occurs when the session attaches to a database by PATHNAME. For example:

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

This error occurs because the CASCADE option is ignored because the Oracle CDD/Repository does not support CASCADE. The workaround is to attach by FILENAME and perform the metadata operation.

In a future release of Oracle Rdb, an informational message will be issued describing the downgrade from CASCADE to RESTRICT in such cases.

7.2.3 Problem Exporting and Importing Sequences with ANSI−Style Databases

Exporting and importing sequences defined in an ANSI−style databases may result in an error. An error will occur if a sequence exists in the database with another object imported after the sequence. For example, importing an ANSI−style database which has sequences and modules defined will return an error. For example:

```
%SQL-F-BADCORATT, invalid core attribute 00, 14 in .RBR file
```

This problem will be fixed in a future release of Oracle Rdb.

7.2.4 System Relation Change for International Database Users

Due to an error in creating the RDB$FIELD_VERSIONS system relation, another system relation, RDB$STORAGE_MAP_AREAS, cannot be accessed if the session character sets are not set to DEC_MCS.

This problem prevents the new Oracle Rdb GUIs, specifically the Oracle Rdb Schema Manager, from viewing indexes and storage maps from existing Oracle Rdb databases.

The problem can be easily corrected by executing the following SQL statement after attaching to the database:

```
SQL> UPDATE RDB$FIELD_VERSIONS SET RDB$FIELD_SUB_TYPE = 32767
cont> WHERE RDB$FIELD_NAME = 'RDB$AREA_NAME';
```

7.2.5 Single Statement CALL Does Not Support Truncated Parameter List or DEFAULT Keyword

Oracle Rdb now allows the CALL statement in a compound statement to omit trailing IN mode parameters which have had a DEFAULT value defined in the procedure definition. Also supported is the DEFAULT keyword to replace an explicit value for the parameter.

However, the simple CALL statement (used outside a BEGIN END block) is not adaptable in this way and requires a full set of parameters and values. This is because a parameter signature is calculated for this type of CALL statement so that the parameter block passed by the calling routine and used by the called routine match exactly in parameter count and data types.

This is a permanent restriction for the simple CALL statement.

The following example shows that truncated parameter lists are fully supported by the compound use form of the CALL statement, but not by the simple CALL statement.

```
SQL> ATTACH 'FILENAME db$:scratch';
SQL> CREATE MODULE mmm
cont> PROCEDURE mmm_p (IN :a INTEGER DEFAULT 0, IN :b INTEGER DEFAULT 1);
cont> TRACE :a, :b;
cont> END MODULE;
SQL> SET FLAGS 'Trace';
SQL> CALL mmm_p (10,20);
~Xt: 10         20
SQL> CALL mmm_p (10);
%SQL-F-ARGCOUNT, Procedure MMM_P expected 2 parameters, was passed 1
SQL> call MMM_P ();
%SQL-F-ARGCOUNT, Procedure MMM_P expected 2 parameters, was passed 0
SQL> begin
cont> CALL mmm_p (10,20);
cont> CALL mmm_p (10);
cont> call mmm_p ();
cont> END;
~Xt: 10         20
~Xt: 10         1
~Xt: 0          1
```

For maximum flexibility, use the CALL statement inside a compound statement which supports truncated parameter lists, the DEFAULT keyword, and full value expressions for parameter arguments.

7.2.6 Single Statement LOCK TABLE is Not Supported for SQL Module Language and SQL Precompiler

The new LOCK TABLE statement is not currently supported as a single statement within the module language or embedded SQL language compiler.

Instead you must enclose the statement in a compound statement. That is, use BEGIN... END around the statement as shown in the following example. This format provides all the syntax and flexibility of LOCK TABLE.

This restriction does not apply to interactive or dynamic SQL.

The following extract from the module language listing file shows the reported error if you use LOCK TABLE as a single statement procedure. The other procedure in the same module is acceptable because it uses a compound statement that contains the LOCK TABLE statement.

```
1 MODULE sample_test
2 LANGUAGE C
3 PARAMETER COLONS
4
5 DECLARE ALIAS FILENAME 'mf_personnel'
6
7 PROCEDURE a (SQLCODE);
8 LOCK TABLE employees FOR EXCLUSIVE WRITE MODE;
%SQL-F-WISH_LIST, (1) Feature not yet implemented - LOCK TABLE requires compound
statement
9
10 PROCEDURE b (SQLCODE);
11 BEGIN
12 LOCK TABLE employees FOR EXCLUSIVE WRITE MODE;
13 END;
```

To workaround this problem of using LOCK TABLE for SQL module language or embedded SQL application, use a compound statement in an EXEC SQL statement.

7.2.7 Restriction for CREATE STORAGE MAP Statement on Table with Data

Oracle Rdb V7.0 added support that allows a storage map to be added to an existing table that contains data. The Oracle Rdb7 Guide to Database Design and Definition describes this feature and lists restrictions.

Oracle Rdb release 7.1 adds the restriction that the storage map cannot include a WITH LIMIT clause for the storage area. The following example shows the resulting error:

```
SQL> create table MAP_TEST1 (a integer, b char(10));
SQL> create index MAP_TEST1_INDEX on MAP_TEST1 (a);
SQL> insert into MAP_TEST1 (a, b) values (3, 'Third');
1 row inserted
SQL> create storage map MAP_TEST1_MAP for MAP_TEST1
cont> store using (a) in RDB$SYSTEM
```

```
cont>     with limit of (10);               -- cannot use WITH LIMIT clause
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-RELNOTEMPTY, table "MAP_TEST1" has data in it
-RDMS-E-NOCMPLXMAP, can not use complex map for non-empty table
```

7.2.8 Multistatement or Stored Procedures May Cause Hangs

Long−running multistatement or stored procedures can cause other users in the database to hang if the procedures obtain resources needed by those other users. Some resources obtained by the execution of a multistatement or stored procedure are not released until the multistatement or stored procedure finishes. Thus, any−long running multistatement or stored procedure can cause other processes to hang. This problem can be encountered even if the statement contains SQL COMMIT or ROLLBACK statements.

The following example demonstrates the problem. The first session enters an endless loop; the second session attempts to backup the database but hangs forever.

```
Session 1:

SQL> attach 'filename MF_PERSONNEL';
SQL> create function LIB$WAIT (in real by reference)
cont> returns integer;
cont> external name LIB$WAIT location 'SYS$SHARE:LIBRTL.EXE'
cont> language general general parameter style variant;
SQL> commit;
        .
        .
        .
$ SQL
SQL> attach 'filename MF_PERSONNEL';
SQL> begin
cont> declare :LAST_NAME LAST_NAME_DOM;
cont> declare :WAIT_STATUS integer;
cont> loop
cont> select LAST_NAME into :LAST_NAME
cont> from EMPLOYEES where EMPLOYEE_ID = '00164';
cont> rollback;
cont> set :WAIT_STATUS = LIBWAIT (5.0);
cont> set transaction read only;
cont> end loop;
cont> end;

Session 2:

$ RMU/BACKUP/LOG/ONLINE MF_PERSONNEL MF_PERSONNEL

From a third session, you can see that the backup process is waiting
for a lock held in the first session:

$ RMU/SHOW LOCKS /MODE=BLOCKING MF_PERSONNEL
    .
    .
    .
Resource: nowait signal

ProcessID Process Name        Lock ID  System ID Requested Granted
--------- ---------------     --------- -------- --------- -------
20204383  RMU BACKUP.....     5600A476  00010001 CW        NL
```

```
2020437B  SQL............     3B00A35C  00010001 PR       PR
```

There is no workaround for this restriction. When the multistatement or stored procedure finishes execution, the resources needed by other processes are released.

7.2.9 Use of Oracle Rdb from Shareable Images

If code in the image initialization routine of a shareable image makes any calls into Oracle Rdb, through SQL or any other means, access violations or other unexpected behavior may occur if Oracle Rdb images have not had a chance to do their own initialization.

To avoid this problem, applications must take one of the following steps:

- Do not make Oracle Rdb calls from the initialization routines of shareable images.
- Link in such a way that the RDBSHR.EXE image initializes first. You can do this by placing the reference to RDBSHR.EXE and any other Oracle Rdb shareable images last in the linker options file.

This is not a bug; it is a restriction resulting from the way OpenVMS image activation works.

7.3 Oracle RMU Known Problems and Restrictions

This section describes known problems and restrictions for the RMU interface for release 7.1.

7.3.1 RMU/CONVERT Fails to Correctly Define the RDB$WORKLOAD Table

When a database is converted to Rdb 7.1 and the optional system table RDB$WORKLOAD is present, Rdb fails to correctly define the metadata for this table, and SQL is unable to see the data type for the RDB$NULL_FACTOR column.

The collection and utilization of workload data is unaffected by this problem. Only SQL applications are affected.

The following is an example of a database incorrectly converted from Rdb 7.0 to Rdb 7.1:

```
SQL> show table rdb$workload
Information for table RDB$WORKLOAD

Columns for table RDB$WORKLOAD:
Column Name                    Data Type        Domain
-----------                    ---------        ------
RDB$CREATED                    DATE VMS
RDB$LAST_ALTERED               DATE VMS
RDB$DUPLICITY_FACTOR           BIGINT(7)
RDB$NULL_FACTOR                Data type: 0
RDB$RELATION_ID                INTEGER
RDB$FLAGS                      INTEGER
RDB$FIELD_GROUP                CHAR(31)
RDB$SECURITY_CLASS             CHAR(20)
```

The RDB$NULL_FACTOR datatype is incorrectly interpreted. This will result in the following problem:

```
SQL> select rdb$null_factor from rdb$workload;
%SQL-F-FLDNOTCRS, Column RDB$NULL_FACTOR was not found in the tables in current
scope
```

A workaround for this problem is to have a sufficiently privileged user execute the following SQL command, commit, and then have applications that use this column DISCONNECT and reattach to the database.

```
SQL> update rdb$relation_fields set rdb$field_source='RDB$SCALED_COUNTER'
cont> where rdb$field_source='RDB$PROBABILITY';
```

This problem will be corrected in Oracle Rdb Release 7.1.0.2.

7.3.2 RMU Convert Fails When Maximum Relation ID is Exceeded

If, when relation IDs are assigned to new system tables during an RMU Convert of an Oracle Rdb V7.0 database to a V7.1 database, the maximum relation ID of 8192 allowed by Oracle Rdb is exceeded, the fatal error %RMU−F−RELMAXIDBAD is displayed and the database is rolled back to V70. Contact your Oracle

support representative if you get this error. Note that when the database is rolled back, the fatal error %RMU−F−CVTROLSUC is displayed to indicate that the rollback was successful but caused by the detection of a fatal error and not requested by the user.

This condition only occurs if there are an extremely large number of tables defined in the database or if a large number of tables were defined but have subsequently been deleted.

The following example shows both the %RMU−F−RELMAXIDBAD error message if the allowed database relation ID maximum of 8192 is exceeded and the %RMU−F−CVTROLSUC error message when the database has been rolled back to V7.0 since it cannot be converted to V7.1:

```
$rmu/convert mf_personnel
%RMU−I−RMUTXT_000, Executing RMU for Oracle Rdb V7.1−00
Are you satisfied with your backup of
 DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1 and your backup of
 any associated .aij files [N]? Y
 %RMU−I−LOGCONVRT, database root converted to current structure level
 %RMU−F−RELMAXIDBAD, ROLLING BACK CONVERSION − Relation ID exceeds maximum
  8192 for system table RDB$RELATIONS
 %RMU−F−CVTROLSUC, CONVERT rolled−back for
  DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1 to version V7.0
```

The following example shows the normal case when the maximum allowed relation ID is not exceeded:

```
$rmu/convert mf_personnel
%RMU−I−RMUTXT_000, Executing RMU for Oracle Rdb V7.1−00
Are you satisfied with your backup of
 DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1 and your backup of
 any associated .aij files [N]? Y
%RMU−I−LOGCONVRT, database root converted to current structure level
%RMU−S−CVTDBSUC, database DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1
 successfully converted from version V7.0 to V7.1
%RMU−I−CVTCOMSUC, CONVERT committed for
 DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1 to version V7.1
```

7.3.3 RMU Unload /After_Journal Requires Accurate AIP Logical Area Information

The RMU Unload /After_Journal command uses the on−disk area inventory pages (AIPs) to determine the appropriate type of each logical area when reconstructing logical dbkeys for records stored in mixed−format storage areas. However, the logical area type information in the AIP is generally unknown for logical areas created prior to Oracle Rdb release 7.0.1. If the RMU Unload /After_Journal command cannot determine the logical area type for one or more AIP entries, a warning message is displayed for each such area and may ultimately return logical dbkeys with a 0 (zero) area number for records stored in mixed−format storage areas.

In order to update the on−disk logical area type in the AIP, the RMU Repair utility must be used. The INITIALIZE=LAREA_PARAMETERS=optionfile qualifier option file can be used with the TYPE qualifier. For example, to repair the EMPLOYEES table of the MF_PERSONNEL database, you would create an options file that contains the following line:

EMPLOYEES /TYPE=TABLE

For partitioned logical areas, the AREA=name qualifier can be used to identify the specific storage areas that are to be updated. For example, to repair the EMPLOYEES table of the MF_PERSONNEL database for the EMPID_OVER storage area only, you would create an options file that contains the following line:

EMPLOYEES /AREA=EMPID_OVER /TYPE=TABLE

The TYPE qualifier specifies the type of a logical area. The following keywords are allowed:

- TABLE
  Specifies that the logical area is a data table. This would be a table created using the SQL CREATE TABLE syntax.
- B−TREE
  Specifies that the logical area is a B−tree index. This would be an index created using the SQL CREATE INDEX TYPE IS SORTED syntax.
- HASH
  Specifies that the logical area is a hash index. This would be an index created using the SQL CREATE INDEX TYPE IS HASHED syntax.
- SYSTEM
  Specifies that the logical area is a system record that is used to identify hash buckets. Users cannot explicitly create these types of logical areas.

  Note

  This type should NOT be used for the RDB$SYSTEM logical areas. This type does NOT identify system relations.

- BLOB
  Specifies that the logical area is a BLOB repository.

There is no explicit error checking of the type specified for a logical area. However, an incorrect type may cause the RMU Unload /After_Journal command to be unable to correctly return valid, logical dbkeys.

7.3.4 Do Not Use HYPERSORT with RMU Optimize After_Journal Command

The OpenVMS Alpha V7.1 operating system introduced the high−performance Sort/Merge utility (also known as HYPERSORT). This utility takes advantage of the OpenVMS Alpha architecture to provide better performance for most sort and merge operations.

The high−performance Sort/Merge utility supports a subset of the SOR routines. Unfortunately, the high−performance Sort/Merge utility does not support several of the interfaces used by the RMU Optimize After_Journal command. In addition, the high−performance Sort/Merge utility reports no error or warning when being called with the unsupported options used by the RMU Optimize After_Journal command.

Because of this, the use of the high−performance Sort/Merge utility is not supported for the RMU Optimize After_Journal command. Do not define the logical name SORTSHR to reference HYPERSORT.EXE.

7.3.5 Changes in EXCLUDE and INCLUDE Qualifiers for RMU Backup

The RMU Backup command no longer accepts both the Include and Exclude qualifiers in the same command. This change removes the confusion over exactly what gets backed up when Include and Exclude are specified on the same line, but does not diminish the capabilities of the RMU Backup command.

To explicitly exclude some storage areas from a backup, use the Exclude qualifier to name the storage areas to be excluded. This causes all storage areas to be backed up except for those named by the Exclude qualifier.

Similarly, the Include qualifier causes only those storage areas named by the qualifier to be backed up. Any storage area not named by the Include qualifier is not backed up. The Noread_only and Noworm qualifiers continue to cause read−only storage areas and WORM storage areas to be omitted from the backup even if these areas are explicitly listed by the Include qualifier.

Another related change is in the behavior of EXCLUDE=*. In previous versions, EXCLUDE=* caused all storage areas to be backed up. Beginning with V7.1, EXCLUDE=* causes only a root backup to be done. A backup created by using EXCLUDE=* can be used only by the RMU Restore Only_Root command.

7.3.6 Default for RMU CRC Qualifier Changing in Future Release

The default behavior for the Crc qualifier for the following RMU commands is changing in a future release of Oracle Rdb:

- Backup
- Backup After_Journal
- Backup Plan
- Optimize After_Journal

Currently, the default value for the CRC qualifier is:

- Crc=Autodin_II is the default for NRZ/PE (800/1600 bits/inch) tape drives
- Crc=Checksum is the default for GCR (6250 bits/inch) tape drives and for TA78, TA79, and TA81 tape drives
- Nocrc is the default for TA90 (IBM 3480 class) drives

In a future release, the default value for the CRC qualifier will be Crc=Checksum for all tape drives except NRZ/PE (800/1600 bits/inch) tape drives. The default qualifier for the NRZ/PE (800/1600 bits/inch) tape drives will remain Crc=Autodin_II. The Crc=Checksum qualifier verifies the checksum on each buffer of data before it is written to tape or disk. This provides end−to−end error detection for the backup file I/O.

Oracle Corporation recommends that you accept the new behavior, that will be the default in a future release of Oracle Rdb, for your applications. The default behavior prevents you from including corrupt database pages in backup files and optimized .aij files. Without the checksum verifications, corrupt data pages in these files are not detected when the files are restored. The corruptions on the restored page may not be detected until weeks or months after the backup file is created, or it is possible the corruption may not be detected at all.

7.3.7 RMU Backup Operations Should Use Only One Type of Tape Drive

When using more than one tape drive for an RMU Backup command, all of the tape drives must be of the same type (for example, all the tape drives must be TA90s or TZ87s or TK50s). Using different tape drive types (for example, one TK50 and one TA90) for a single database backup operation may make database restoration difficult or impossible.

Oracle RMU attempts to prevent using different tape drive densities during a backup operation, but is not able to detect all invalid cases and expects that all tape drives for a backup are of the same type.

As long as all of the tapes used during a backup operation can be read by the same type of tape drive during a restore operation, the backup is likely valid. This may be the case, for example, when using a TA90 and a TA90E.

Oracle Corporation recommends that, on a regular basis, you test your backup and recovery procedures and environment using a test system. You should restore the database and then recover using AIJs to simulate failure recovery of the production system.

Consult the Oracle Rdb7 Guide to Database Maintenance, the Oracle Rdb7 Guide to Database Design and Definition, and the Oracle RMU Reference Manual for additional information about Oracle Rdb backup and restore operations.

7.3.8 RMU/VERIFY Reports PGSPAMENT or PGSPMCLST Errors

RMU/VERIFY may sometimes report PGSPAMENT or PGSPMCLST errors when verifying storage areas. These errors indicate that the Space Area Management (SPAM) page fullness threshold for a particular data page does not match the actual space usage on the data page. For a further discussion of SPAM pages, consult the Oracle Rdb7 Guide to Database Maintenance.

In general, these errors will not cause any adverse affect on the operation of the database. There is potential for space on the data page to not be totally utilized, or for a small amount of extra I/O to be expended when searching for space in which to store new rows. But unless there are many of these errors then the impact should be negligible.

It is possible for these inconsistencies to be introduced by errors in Oracle Rdb. When those cases are discovered, Oracle Rdb is corrected to prevent the introduction of the inconsistencies. It is also possible for these errors to be introduced during the normal operation of Oracle Rdb. The following scenario can leave the SPAM pages inconsistent:

1. A process inserts a row on a page, and updates the threshold entry on the corresponding SPAM page to reflect the new space utilization of the data page. The data page and SPAM pages are not flushed to disk.
2. Another process notifies the first process that it would like to access the SPAM page being held by the process. The first process flushes the SPAM page changes to disk and releases the page. Note that it has not flushed the data page.
3. The first process then terminates abnormally (for example, from the DCL STOP/IDENTIFICATION command). Since that process never flushed the data page to disk, it never wrote the changes to the Recovery Unit Journal (RUJ) file. Since there were no changes in the RUJ file for that data page then

the Database Recovery (DBR) process did not need to roll back any changes to the page. The SPAM page retains the threshold update change made above even though the data page was never flushed to disk.

While it would be possible to create mechanisms to ensure that SPAM pages do not become out of synch with their corresponding data pages, the performance impact would not be trivial. Since these errors are relatively rare and the impact is not significant, then the introduction of these errors is considered to be part of the normal operation of Oracle Rdb. If it can be proven that the errors are not due to the scenario above, then Oracle Product Support should be contacted.

PGSPAMENT and PGSPMCLST errors may be corrected by doing any one of the following operations:

- Recreate the database by performing:
    1. SQL EXPORT
    2. SQL DROP DATABASE
    3. SQL IMPORT
- Recreate the database by performing:
    1. RMU/BACKUP
    2. SQL DROP DATABASE
    3. RMU/RESTORE
- Repair the SPAM pages by using the RMU/REPAIR command. Note that the RMU/REPAIR command does not write its changes to an after−image journal (AIJ) file. Therefore, Oracle recommends that a full database backup be performed immediately after using the RMU/REPAIR command.

7.4 Known Problems and Restrictions in All Interfaces for Release 7.0 and Earlier

The following problems and restrictions from release 7.0 and earlier still exist.

7.4.1 Converting Single−File Databases

Because of a substantial increase in the database root file information for V7.0, you should ensure that you have adequate disk space before you use the RMU Convert command with single−file databases and V7.0 or higher.

The size of the database root file of any given database increases a minimum of 13 blocks and a maximum of 597 blocks. The actual increase depends mostly on the maximum number of users specified for the database.

7.4.2 Row Caches and Exclusive Access

If a table has a row−level cache defined for it, the Row Cache Server (RCS) may acquire a shared lock on the table and prevent any other user from acquiring a Protective or Exclusive lock on that table.

7.4.3 Exclusive Access Transactions May Deadlock with RCS Process

If a table is frequently accessed by long running transactions that request READ/WRITE access reserving the table for EXCLUSIVE WRITE and if the table has one or more indexes, you may experience deadlocks between the user process and the Row Cache Server (RCS) process.

There are at least three suggested workarounds to this problem:

- Reserve the table for SHARED WRITE
- Close the database and disable row cache for the duration of the exclusive transaction
- Change the checkpoint interval for the RCS process to a time longer than the time required to complete the batch job and then trigger a checkpoint just before the batch job starts. Set the interval back to a smaller interval after the checkpoint completes.

7.4.4 Strict Partitioning May Scan Extra Partitions

When you use a WHERE clause with the less than (<) or greater than (>) operator and a value that is the same as the boundary value of a storage map, Oracle Rdb scans extra partitions. A boundary value is a value specified in the WITH LIMIT OF clause. The following example, executed while the logical name RDMS$DEBUG_FLAGS is defined as "S", illustrates the behavior:

```
ATTACH 'FILENAME MF_PERSONNEL';
CREATE TABLE T1 (ID INTEGER, LAST_NAME CHAR(12), FIRST_NAME CHAR(12));
CREATE STORAGE MAP M FOR T1 PARTITIONING NOT UPDATABLE
 STORE USING (ID)
 IN EMPIDS_LOW WITH LIMIT OF (200)
```

```
 IN EMPIDS_MID WITH LIMIT OF (400)
 OTHERWISE IN EMPIDS_OVER;
INSERT INTO T1 VALUES (150,'Boney','MaryJean');
INSERT INTO T1 VALUES (350,'Morley','Steven');
INSERT INTO T1 VALUES (300,'Martinez','Nancy');
INSERT INTO T1 VALUES (450,'Gentile','Russ');
SELECT * FROM T1 WHERE ID > 400;
Conjunct Get Retrieval sequentially of relation T1
Strict Partitioning: part 2 3
ID LAST_NAME FIRST_NAME
450 Gentile Russ
1 row selected
```

In the previous example, partition 2 does not need to be scanned. This does not affect the correctness of the result. Users can avoid the extra scan by using values other than the boundary values.

7.4.5 Restriction When Adding Storage Areas with Users Attached to Database

If you try to interactively add a new storage area where the page size is less than the existing page size and the database has been manually opened or users are active, the add operation fails with the following error:

```
%RDB-F-SYS_REQUEST, error from system services request
-RDMS-F-FILACCERR, error opening database root DKA0:[RDB]TEST.RDB;1
-SYSTEM-W-ACCONFLICT, file access conflict
```

You can make this change only when no users are attached to the database and, if the database is set to OPEN IS MANUAL, the database is closed. Several internal Oracle Rdb data structures are based on the minimum page size and these structures cannot be resized if users are attached to the database.

Furthermore, because this particular change is not recorded in the AIJ, any recovery scenario fails. Note also that if you use .aij files, you must backup the database and restart after-image journaling because this change invalidates the current AIJ recovery.

7.4.6 Support for Single-File Databases to Be Dropped in a Future Release

Oracle Rdb currently supports both single-file and multifile databases on all platforms. However, single-file databases will not be supported in a future release of Oracle Rdb. At that time, Oracle Rdb will provide the means to easily convert single-file databases to multifile databases.

Oracle Rdb recommends that users with single-file databases perform the following actions:

- Use the Oracle RMU commands, such as Backup and Restore, to make copies, backup, or move single-file databases. Do not use operating system commands to copy, back up, or move databases.
- Create new databases as multifile databases even though single-file databases are supported.

7.4.7 Multiblock Page Writes May Require Restore Operation

If a node fails while a multiblock page is being written to disk, the page in the disk becomes inconsistent, and is detected immediately during failover. (Failover is the recovery of an application by restarting it on another computer.) The problem is rare, and occurs because only single−block I/O operations are guaranteed by OpenVMS to be written atomically. This problem has never been reported by any customer and was detected only during stress tests in our labs.

Correct the page by an area−level restore operation. Database integrity is not compromised, but the affected area is not available until the restore operation completes.

A future release of Oracle Rdb will provide a solution that guarantees multiblock atomic write operations. Cluster failovers will automatically cause the recovery of multiblock pages, and no manual intervention will be required.

7.4.8 Network Link Failure Does Not Allow DISCONNECT to Clean Up Transactions

If a program attaches to a database on a remote node and it loses the connection before the COMMIT statement is issued, there is nothing you can do except exit the program and start again.

The problem occurs when a program is connected to a remote database and updates the database, but then just before it commits, the network fails. When the commit executes, SQL shows, as it normally should, that the program has lost the link. Assume that the user waits for a minute or two, then tries the transaction again. The problem is that when the start transaction is issued for the second time, it fails because old information still exists about the previous failed transaction. This occurs even if the user issues a DISCONNECT statement (in V4.1 and earlier, a FINISH statement), which also fails with an RDB−E−IO_ERROR error message.

7.4.9 Replication Option Copy Processes Do Not Process Database Pages Ahead of an Application

When a group of copy processes initiated by the Replication Option (formerly Data Distributor) begins running after an application has begun modifying the database, the copy processes catch up to the application and are not able to process database pages that are logically ahead of the application in the RDB$CHANGES system relation. The copy processes all align waiting for the same database page and do not move on until the application has released it. The performance of each copy process degrades because it is being paced by the application.

When a copy process completes updates to its respective remote database, it updates the RDB$TRANSFERS system relation and then tries to delete any RDB$CHANGES rows not needed by any transfers. During this process, the RDB$CHANGES table cannot be updated by any application process, holding up any database updates until the deletion process is complete. The application stalls while waiting for the RDB$CHANGES table. The resulting contention for RDB$CHANGES SPAM pages and data pages severely impacts performance throughput, requiring user intervention with normal processing.

This is a known restriction in V4.0 and higher. Oracle Rdb uses page locks as latches. These latches are held only for the duration of an action on the page and not to the end of transaction. The page locks also have blocking asynchronous system traps (ASTs) associated with them. Therefore, whenever a process requests a page lock, the process holding that page lock is sent a blocking AST (BLAST) by OpenVMS. The process that receives such a blocking AST queues the fact that the page lock should be released as soon as possible. However, the page lock cannot be released immediately.

Such work requests to release page locks are handled at verb commit time. An Oracle Rdb verb is an Oracle Rdb query that executes atomically, within a transaction. Therefore, verbs that require the scan of a large table, for example, can be quite long. An updating application does not release page locks until its verb has completed.

The reasons for holding on to the page locks until the end of the verb are fundamental to the database management system.

7.5 SQL Known Problems and Restrictions for Oracle Rdb Release 7.0 and Earlier

The following problems and restrictions from Oracle Rdb Release 7.0 and earlier still exist.

7.5.1 SQL Does Not Display Storage Map Definition After Cascading Delete of Storage Area

When you drop a storage area using the CASCADE keyword and that storage area is not the only area to which the storage map refers, the SHOW STORAGE MAP statement no longer shows the placement definition for that storage map.

The following example demonstrates this restriction:

```
SQL> SHOW STORAGE MAP DEGREES_MAP1
     DEGREES_MAP1
 For Table:             DEGREES1
 Compression is:        ENABLED
 Partitioning is:       NOT UPDATABLE
 Store clause:          STORE USING (EMPLOYEE_ID)
            IN DEG_AREA WITH LIMIT OF ('00250')
             OTHERWISE IN DEG_AREA2

SQL> DISCONNECT DEFAULT;
SQL> -- Drop the storage area, using the CASCADE keyword.
SQL> ALTER DATABASE FILENAME MF_PERSONNEL
cont> DROP STORAGE AREA DEG_AREA CASCADE;
SQL> -- Display the storage map definition.
SQL> ATTACH 'FILENAME MF_PERSONNEL';
SQL> SHOW STORAGE MAP DEGREES_MAP1
DEGREES_MAP1 For Table: DEGREES1
Compression is: ENABLED
Partitioning is: NOT UPDATABLE
```

The other storage area, DEG_AREA2, still exists, even though the SHOW STORAGE MAP statement does not display it.

A workaround is to use the RMU Extract command with the Items=Storage_Map qualifier to see the mapping.

7.5.2 ARITH_EXCEPT or Incorrect Results Using LIKE IGNORE CASE

When you use LIKE...IGNORE CASE, programs linked under Oracle Rdb V4.2 and V5.1, but run under higher versions of Oracle Rdb, may result in incorrect results or %RDB−E−ARITH_EXCEPT exceptions.

To work around the problem, avoid using IGNORE CASE with LIKE or recompile and relink under a higher version (V6.0 or higher.)

7.5.3 Different Methods of Limiting Returned Rows from Queries

You can establish the query governor for rows returned from a query by using either the SQL SET QUERY LIMIT statement or a logical name. This note describes the differences between the two mechanisms.

If you define the RDMS$BIND_QG_REC_LIMIT logical name to a small value, the query often fails with no rows returned regardless of the value assigned to the logical. The following example demonstrates setting the limit to 10 rows and the resulting failure:

```
$ DEFINE RDMS$BIND_QG_REC_LIMIT 10
$ SQL$
SQL> ATTACH 'FILENAME MF_PERSONNEL';
SQL> SELECT EMPLOYEE_ID FROM EMPLOYEES;
%RDB-F-EXQUOTA, Oracle Rdb runtime quota exceeded
-RDMS-E-MAXRECLIM, query governor maximum limit of rows has been reached
```

Interactive SQL must load its metadata cache for the table before it can process the SELECT statement. In this example, interactive SQL loads its metadata cache to allow it to check that the column EMPLOYEE_ID really exists for the table. The queries on the Oracle Rdb system relations RDB$RELATIONS and RDB$RELATION_FIELDS exceed the limit of rows.

Oracle Rdb does not prepare the SELECT statement, let alone execute it. Raising the limit to a number less than 100 (the cardinality of EMPLOYEES) but more than the number of columns in EMPLOYEES (that is, the number of rows to read from the RDB$RELATION_FIELDS system relation) is sufficient to read each column definition.

To see an indication of the queries executed against the system relations, define the RDMS$DEBUG_FLAGS logical name as "S" or "B".

If you set the row limit using the SQL SET QUERY statement and run the same query, it returns the number of rows specified by the SQL SET QUERY statement before failing:

```
SQL> ATTACH 'FILENAME MF_PERSONNEL';
SQL> SET QUERY LIMIT ROWS 10;
SQL> SELECT EMPLOYEE_ID FROM EMPLOYEES;
EMPLOYEE_ID
00164
00165
   .
   .
   .
00173
%RDB-E-EXQUOTA, Oracle Rdb runtime quota exceeded
-RDMS-E-MAXRECLIM, query governor maximum limit of rows has been reached
```

The SET QUERY LIMIT specifies that only user queries be limited to 10 rows. Therefore, the queries used to load the metadata cache are not restricted in any way.

Like the SET QUERY LIMIT statement, the SQL precompiler and module processor command line qualifiers (QUERY_MAX_ROWS and SQLOPTIONS=QUERY_MAX_ROWS) only limit user queries.

Keep the differences in mind when limiting returned rows using the logical name RDMS$BIND_QG_REC_LIMIT. They may limit more queries than are obvious. This is important when using 4GL tools, the SQL precompiler, the SQL module processor, and other interfaces that read the Oracle Rdb system relations as part of query processing.

7.5.4 Suggestions for Optimal Use of SHARED DATA DEFINITION Clause for Parallel Index Creation

The CREATE INDEX process involves the following steps:

1. Process the metadata.
2. Lock the index name.
   Because new metadata (which includes the index name) is not written to disk until the end of the index process, Oracle Rdb must ensure index name uniqueness across the database during this time by taking a special lock on the provided index name.
3. Read the table for sorting by selected index columns and ordering.
4. Sort the key data.
5. Build the index (includes partitioning across storage areas).
6. Write new metadata to disk.

Step 6 is the point of conflict with other index definers because the system relation and indexes are locked like any other updated table.

Multiple users can create indexes on the same table by using the RESERVING table_name FOR SHARED DATA DEFINITION clause of the SET TRANSACTION statement. For optimal usage of this capability, Oracle Rdb suggests the following guidelines:

- You should commit the transaction immediately after the CREATE INDEX statement so that locks on the table are released. This avoids lock conflicts with other index definers and improves overall concurrency.
- By assigning the location of the temporary sort work files SORTWORK0, SORTWORK1, ... , SORTWORK9 to different disks for each parallel process that issues the SHARED DATA DEFINITION statement, you can increase the efficiency of sort operations. This minimizes any possible disk I/O bottlenecks and allows overlap of the SORT read/write cycle.
- If possible, enable global buffers and specify a buffer number large enough to hold a sufficient amount of table data. However, do not define global buffers larger than the available system physical memory. Global buffers allow sharing of database pages and thus result in disk I/O savings. That is, pages are read from disk by one of the processes and then shared by the other index definers for the same table, reducing the I/O load on the table.
- If global buffers are not used, ensure that enough local buffers exist to keep much of the index cached (use the RDM$BIND_BUFFERS logical name or the NUMBER OF BUFFERS IS clause in SQL to change the number of buffers).
- To distribute the disk I/O load, store the storage areas for the indexes on separate disk drives. Note that using the same storage area for multiple indexes results in contention during the index creation (Step 5) for SPAM pages.
- Consider placing the .ruj file for each parallel definer on its own disk or an infrequently used disk.
- Even though snapshot I/O should be minimal, consider disabling snapshots during parallel index creation.
- Refer to the Oracle Rdb7 Guide to Database Performance and Tuning to determine the appropriate working set values for each process to minimize excessive paging activity. In

particular, avoid using working set parameters where the difference between WSQUOTA and WSEXTENT is large. The SORT utility uses the difference between these two values to allocate scratch virtual memory. A large difference (that is, the requested virtual memory grossly exceeds the available physical memory) may lead to excessive page faulting.

- The performance benefits of using SHARED DATA DEFINITION can best be observed when creating many indexes in parallel. The benefit is in the average elapsed time, not in CPU or I/O usage. For example, when two indexes are created in parallel using the SHARED DATA DEFINITION clause, the database must be attached twice, and the two attaches each use separate system resources.
- Using the SHARED DATA DEFINITION clause on a single−file database or for indexes defined in the RDB$SYSTEM storage area is not recommended.

The following table displays the elapsed time benefit when creating multiple indexes in parallel with the SHARED DATA DEFINITION clause. The table shows the elapsed time for ten parallel process index creations (Index1, Index2, ... Index10) and one process with ten sequential index creations (All10). In this example, global buffers are enabled and the number of buffers is 500. The longest time for a parallel index creation is Index7 with an elapsed time of 00:02:34.64, compared to creating ten indexes sequentially with an elapsed time of 00:03:26.66. The longest single parallel create index elapsed time is shorter than the elapsed time of creating all ten of the indexes serially.

Table 7−2 Elapsed Time for Index Creations

Index Create Job	Elapsed Time
Index1	00:02:22.50
Index2	00:01:57.94
Index3	00:02:06.27
Index4	00:01:34.53
Index5	00:01:51.96
Index6	00:01:27.57
Index7	00:02:34.64
Index8	00:01:40.56
Index9	00:01:34.43
Index10	00:01:47.44
All10	00:03:26.66

7.5.5 Side Effect When Calling Stored Routines

When calling a stored routine, you must not use the same routine to calculate argument values by a stored function. For example, if the routine being called is also called by a stored function during the calculation of an argument value, passed arguments to the routine may be incorrect.

The following example shows a stored procedure P being called during the calculation of the arguments for another invocation of the stored procedure P:

```
SQL> create module M
cont>     language SQL
cont>
```

```
cont>     procedure P (in :a integer, in :b integer, out :c integer);
cont>     begin
cont>     set :c = :a + :b;
cont>     end;
cont>
cont>     function F () returns integer
cont>     comment is 'expect F to always return 2';
cont>     begin
cont>     declare :b integer;
cont>     call P (1, 1, :b);
cont>     trace 'returning ', :b;
cont>     return :b;
cont>     end;
cont> end module;
SQL>
SQL> set flags 'TRACE';
SQL> begin
cont> declare :cc integer;
cont> call P (2, F(), :cc);
cont> trace 'Expected 4, got ', :cc;
cont> end;
~Xt: returning 2
~Xt: Expected 4, got 3
```

The result as shown above is incorrect. The routine argument values are written to the called routine's parameter area before complex expression values are calculated. These calculations may (as in the example) overwrite previously copied data.

The workaround is to assign the argument expression (in this example calling the stored function F) to a temporary variable and pass this variable as the input for the routine. The following example shows the workaround:

```
SQL> begin
cont> declare :bb, :cc integer;
cont> set :bb = F();
cont> call P (2, :bb, :cc);
cont> trace 'Expected 4, got ', :cc;
cont> end;
~Xt: returning 2
~Xt: Expected 4, got 4
```

This problem will be corrected in a future version of Oracle Rdb.

7.5.6 Considerations When Using Holdable Cursors

If your applications use holdable cursors, be aware that after a COMMIT or ROLLBACK statement is executed, the result set selected by the cursor may not remain stable. That is, rows may be inserted, updated, and deleted by other users because no locks are held on the rows selected by the holdable cursor after a commit or rollback occurs. Moreover, depending on the access strategy, rows not yet fetched may change before Oracle Rdb actually fetches them.

As a result, you may see the following anomalies when using holdable cursors in a concurrent user environment:

- If the access strategy forces Oracle Rdb to take a data snapshot, the data read and cached may

be stale by the time the cursor fetches the data.
For example, user 1 opens a cursor and commits the transaction. User 2 deletes rows read by user 1 (this is possible because the read locks are released). It is possible for user 1 to report data now deleted and committed.

- If the access strategy uses indexes that allow duplicates, updates to the duplicates chain may cause rows to be skipped, or even revisited.
  Oracle Rdb keeps track of the dbkey in the duplicate chain pointing to the data that was fetched. However, the duplicates chain could be revised by the time Oracle Rdb returns to using it.

Holdable cursors are a very powerful feature for read−only or predominantly read−only environments. However, in concurrent update environments, the instability of the cursor may not be acceptable. The stability of holdable cursors for update environments will be addressed in future versions of Oracle Rdb.

You can define the logical name RDMS$BIND_HOLD_CURSOR_SNAP to the value 1 to force all hold cursors to fetch the result set into a cached data area. (The cached data area appears as a "Temporary Relation" in the optimizer strategy displayed by the SET FLAGS 'STRATEGY' statement or the RDMS$DEBUG_FLAGS "S" flag.) This logical name helps to stabilize the cursor to some degree.

| Contents