Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Recent Developments in Oracle Rdb

Ian Smith
Consulting Member of Staff
Oracle Corporation
Optimizer Enhancements
Query Rewrite

Ongoing improvements to the query compiler
Query Rewrite: Recap

• New query rewrite engine in V7.3

• Make use of nullability knowledge to simplify query
  – Evaluates known not nullable expressions during query compile
  – Knowledge based on constraints
    • not null not deferrable
    • primary key not deferrable

• Eliminates always TRUE and FALSE predicate branches
  – Also applied to CASE statement WHEN Boolean expressions

• Computes integer and floating literal expressions at compile time

• Eliminate constant values from ORDER BY, DISTINCT, UNION DISTINCT, etc.
Query Rewrite: Date/Time

• Recognize run-time functions and convert to compile-time evaluation

• `CAST(string AS TIMESTAMP(n))`
  `CAST(string AS DATE ANSI)`
  `CAST(string AS DATE VMS)`
  Convert string to binary date/time during compile

• e.g. `CAST('2017-1-1' AS DATE)` replaced with literal `date '2017-1-1'`

• `CAST(integer AS INTERVAL ...)`
  If single field interval-qualifier, replace with interval literal
Query Rewrite: Date/Time

• Recognize date ANSI expressions
  ... where cast(posting_timestamp as date) = date'value'

• Here the query wants all timestamps for a specific date but CAST hides the index column posting_timestamp from the optimizer

• Query rewrite changes it to:
  ... where posting_timestamp >= date'value' and posting_timestamp < date'value' + interval'1' day

• Now the column is exposed in a range query
Query Rewrite: BITMAP SCAN and <>

- When bitmap scan is enabled, transform not equal to range query
- e.g. columnname <> value becomes
columnname < value or columnname > value
Query Rewrite: String comparisons (V7.3.3)

• Expanded expression compare for strings
• Must have identical character sets
• Evaluated during compile time:
  – Equals
  – Not equals
  – Greater than
  – Less than
  – Greater or Equal than
  – Less or Equal than
Aggregate Enhancements

Ongoing improvements to the query compiler
COUNT optimization

• Recap: SORTED RANKED index
• COUNT aggregates
  – COUNT(*)
  – COUNT(ALL column)
  – COUNT(DISTINCT column)
• Solved by using special **count scan** optimization
  – Saves CPU time
• Don’t need list of DBKEYs, just the count
• Computed from leaf node cardinality values
New: COUNT optimization

• Observed that similar savings could be made for SORTED indices
• Must do I/O to duplicate node chains
• Still has huge benefit as count is computed deeper in kernel
• Rather than counting returned DBKEY values
• …some tests show 15% lower CPU usage
• Results will vary based on index and query
Multi-Aggregate Optimization

• Recap: existing index optimizations are:
  – Min key lookup
  – Max key lookup
  – Index counts (sorted), and Index counts lookup (sorted ranked)
  – Index distinct (sorted), and Index distinct lookup (sorted ranked)

• These are applied to:
  – MIN (index-column), MAX (index-column)
  – COUNT(*), COUNT (index-column), COUNT (DISTINCT index-column)

• ...but only applied to if one aggregate function used
Multi-Aggregate Optimization (7.3.2.1)

• Optimizations now applied to all candidate aggregate functions
• Reduces I/O and CPU usage for the query
• Applied even when aggregates combined in expressions
• e.g. Range calculation: MAX (v) – MIN (v)
SQL> select min (salary_amount), max (salary_amount), count (salary_amount) cont> from salary_history;
Tables:
  0 = SALARY_HISTORY
Aggregate: 0:MIN (0.SALARY_AMOUNT) Q2
  1:MAX (0.SALARY_AMOUNT) Q2
  2:COUNT (0.SALARY_AMOUNT) Q2
Index only retrieval of relation 0:SALARY_HISTORY
  Index name  SALARY_NDX [0:0]

$7,000.00  $93,340.00  729
1 row selected
SQL>
SQL> select min (salary_amount), max (salary_amount), count (salary_amount) 
   cont> from salary_history;

Tables:
   0 = SALARY_HISTORY
Aggregate: 0:MIN (0.SALARY_AMOUNT) Q2
   1:MAX (0.SALARY_AMOUNT) Q2
   2:COUNT (0.SALARY_AMOUNT) Q2
Index only retrieval of relation 0:SALARY_HISTORY
   Index name  SALARY_NDX [0:0]   Min key lookup
Index only retrieval of relation 0:SALARY_HISTORY
   Index name  SALARY_NDX [0:0]   Max key lookup
Index only retrieval of relation 0:SALARY_HISTORY
   Index name  SALARY_NDX [0:1]   Index counts
   Keys: NOT MISSING (0.SALARY_AMOUNT)

  $7,000.00    $93,340.00                    729
1 row selected  
SQL>

Uses several optimized scans
DDL
(Data Definition Language)
Changes
Temporary Tables

• Now supports LARGE MEMORY option
• Rows and data structures reside in P2 space (64 bit memory)
• Frees up P0/P1 space for applications
• New clause for:
  – Create local temporary table
  – Create global temporary table
  – Declare local temporary table (V7.3.2)
Changes: TRUNCATE TABLE (V7.3.2)

- The SQL Database Language Standard defines new behavior and new syntax
- `restart identity` and `continue identity` clauses
- Previously `truncate table` would always `restart identity` (reset to the initial value)
- SQL dialects (SQL2011, and future) will default to `continue identity`
New: Truncate Table ... continue identity

```
SQL> truncate table ACTIVITY_LOG continue identity;
SQL>
SQL> show sequence ACTIVITY_LOG;
   ACTIVITY_LOG
Sequence Id: 9
An identity column sequence.
Initial Value: 1
Minimum Value: 1
Maximum Value: (none)
Next Sequence Value: 1001
Increment by: 1
Cache Size: 20
No Order
No Cycle
No Randomize
Wait
Comment: column IDENTITY sequence
SQL>
```
Generated Columns - Background

• Rdb V1.0

• Supported COMPUTED BY columns
  – Virtual
  – Take just 1 in null bit vector in row
  – Always a read-only column

• Evaluated during select, or in where clauses for update and delete statements
Generated Columns

• Rdb V7.1

• Supported AUTOMATIC columns
  – Automatic insert as ...
  – Automatic update as ...
  – Automatic as ...

• Use space in a row

• Data type derived from value expression

• Evaluated on **insert** and/or **update** statements

• Act like read-only columns, but Rdb stores data
Generated Columns - Identity

- Rdb V7.1
- IDENTITY
  - Based on CREATE SEQUENCE feature
  - Special sequence created; same name as table
  - Therefore, at most one per table
  - Looks like AUTOMATIC AS tablename.NEXTVAL
- Uses space in row
- Column can be given a data type (or domain)
- Acts like read-only column; but Rdb stores data
Generated Columns

• Database administrator may want to insert rows which have values for these types of columns
  – Sometimes these columns need correction
  – Might want to reload the table without re-computing values

• Provide SET FLAGS ‘AUTO_OVERRIDE’
  – Allows privileged user (DBADM) to manage updates

• Restructure and reload supported by RMU
  – RMU/LOAD/VIRTUAL_FIELDS=AUTOMATIC
  – SET FLAGS not required in this case
Generated Columns – New Syntax (V7.3.3)

• Based on ANSI/ISO SQL, and Oracle Database language
• Syntax maps to AUTOMATIC and IDENTITY implementation
  – GENERATED ALWAYS AS ( value-expression )
  – GENERATED ALWAYS AS IDENTITY
• New: can specify domain or data type for column
Generated Columns (V7.3.3)

• Also supporting new functionality
  – GENERATED BY DEFAULT AS IDENTITY
  – GENERATED BY DEFAULT AS ( value-expression )

• These types of generated columns can be updated by the application

• If a value is inserted, then no automatic generation will take place

• No privileges or use of AUTO_OVERRIDE required
SQL> create table EMPLOYEES
cont> (badge_num integer generated always as identity (cache 100)
cont> ,last_name varchar(40)
cont> ,first_name varchar(40)
cont> ,created_date generated always as ( current_date )
cont> ,created_by generated by default as ( current_user )
cont> );

The data type is optional. Implicit CAST performed when specified.
Generated Columns - Identity

• Any GENERATED BY DEFAULT AS IDENTITY column can be manually updated
• Inserted/updated values should be within the range of the sequence
• Rdb can no longer guarantee unique values
• Collisions must be handled by the application
Generated Columns – DEFAULT (V7.3.3)

• Now allow **default** keyword assignment to AUTOMATIC, IDENTITY, GENERATED ALWAYS and GENERATED BY DEFAULT columns

SQL> !! These two statement should be equivalent
SQL> insert into SAMPLE1 default values;
1 row inserted
SQL> insert into SAMPLE1 (employee_id, birthday, last_name)
cont>     values (default, default, default);
%RDB-E-READ_ONLYFIELD, attempt to update the read-only field
EMPLOYEE_ID
SQL>
New and Improved Built-in Functions

Oracle Database and ANSI/ISO SQL Language support
TRIM optimizations (V7.3.2)

• TRIM function is based on ANSI/ISO SQL syntax
• Really three separate function
  – TRIM (LEADING ...)
  – TRIM (TRAILING ...)
  – TRIM (BOTH ...)
• Now implemented for most character sets as separate functions
• Reduced code path for normal usage
LTRIM and RTRIM

• Supported in prior versions in the SQL FUNCTIONS library
• Limited character set support
• Fixed size result; always VARCHAR(2000)
• Common usage through Oracle tools and OCI Services for Rdb
LTRIM and RTRIM

• New native support in Oracle Rdb
• Supports Oracle Database null semantics
• Polymorphic (adapts to character set and length)
• Lower overhead in queries
• Similar to TRIM function but trim any of the characters in the trim-string
• Access to older versions will revert to SQL_FUNCTIONS equivalents
  – SQL Module Language or SQL Pre-compiler applications require recompile to use new implementation
LISTAGG

• Allows values in the group to be concatenated to a single value
• Best explained with examples
### LISTAGG

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>SALARY_START</th>
<th>SALARY_END</th>
<th>SALARY_AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00369</td>
<td>1982-06-02</td>
<td>NULL</td>
<td>$57,410.00</td>
</tr>
<tr>
<td>00374</td>
<td>1978-10-23</td>
<td>1979-08-19</td>
<td>$43,310.00</td>
</tr>
<tr>
<td>00374</td>
<td>1979-08-19</td>
<td>1980-01-18</td>
<td>$45,427.00</td>
</tr>
<tr>
<td>00374</td>
<td>1980-01-18</td>
<td>1981-01-12</td>
<td>$47,732.00</td>
</tr>
<tr>
<td>00374</td>
<td>1981-01-12</td>
<td>1981-10-15</td>
<td>$47,919.00</td>
</tr>
<tr>
<td>00374</td>
<td>1981-10-15</td>
<td>1982-10-10</td>
<td>$50,424.00</td>
</tr>
<tr>
<td>00374</td>
<td>1982-10-10</td>
<td>NULL</td>
<td>$50,424.00</td>
</tr>
<tr>
<td>00405</td>
<td>1975-07-01</td>
<td>1976-02-26</td>
<td>$27,554.00</td>
</tr>
</tbody>
</table>

**Group by EMPLOYEE_ID**

- Group 00374
## LISTAGG

The `LISTAGG` function in SQL is used to generate a list of values from a column, separated by a specified character. The syntax for `LISTAGG` is:

```
LISTAGG ( expression [ , expression ] )
```

In the provided example, the `LISTAGG` function is applied to the `SALARY_START` column to create a list of salary start dates:

```
LISTAGG ( salary_start )
```

### Table Data

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>SALARY_START</th>
<th>SALARY_END</th>
<th>SALARY_AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00369</td>
<td>1982-06-02</td>
<td>NULL</td>
<td>$57,410.00</td>
</tr>
<tr>
<td>00374</td>
<td>1978-10-23</td>
<td>1979-08-19</td>
<td>$43,310.00</td>
</tr>
<tr>
<td>00374</td>
<td>1979-08-19</td>
<td>1980-01-18</td>
<td>$45,427.00</td>
</tr>
<tr>
<td>00374</td>
<td>1980-01-18</td>
<td>1981-01-12</td>
<td>$47,732.00</td>
</tr>
<tr>
<td>00374</td>
<td>1981-01-12</td>
<td>1981-10-15</td>
<td>$47,919.00</td>
</tr>
<tr>
<td>00374</td>
<td>1981-10-15</td>
<td>1982-10-10</td>
<td>$50,424.00</td>
</tr>
<tr>
<td>00374</td>
<td>1982-10-10</td>
<td>NULL</td>
<td>$50,424.00</td>
</tr>
<tr>
<td>00405</td>
<td>1975-07-01</td>
<td>1976-02-26</td>
<td>$27,554.00</td>
</tr>
</tbody>
</table>
LISTAGG

• Other aggregates have a known data type and size result
• MEDIAN, MAX, MIN reflect the input values
• AVG, STDDEV, VARIANCE always DOUBLE PRECISION
• COUNT is an integer (int or bigint)

• How long is long is the LISTAGG result?
• Defaults to VARCHAR(4000) result but is user settable
LISTAGG

• Accepts columns from the group
• Can define a separator string for the list
• Includes an ORDER BY clause to arrange ordering of the result string
  – WITHIN GROUP (ORDER BY ...)
• Can use DISTINCT
• Optional clauses to control action on string truncation
  – ON OVERFLOW ERROR
  – ON OVERFLOW TRUNCATE
LISTAGG

• What if group is large or result exceeds output buffer?

```
SELECT department_id,
    LISTAGG(last_name, '; ') WITHIN GROUP (ORDER BY hire_date) AS emps
FROM employees
GROUP BY department_id;
```

%RDB-E-CONCATTRUNC, data exception - string data, right truncation for concatenate
-RDMS-F-TRUNRGTSTR, truncation of right-hand side string for assignment has occurred

separator string
LISTAGG

- This is equivalent to ON OVERFLOW ERROR syntax

```
SELECT department_id,
       LISTAGG(last_name, '; ' ON OVERFLOW ERROR) 
       WITHIN GROUP (ORDER BY hire_date) AS emps 
FROM employees 
GROUP BY department_id;
```

%RDB-E-CONCATTRUNC, data exception - string data, right truncation for concatenate 
-RDMS-F-TRUNRGTSTR, truncation of right-hand side string for assignment has occurred
LISTAGG

- Can control behavior with ON OVERFLOW TRUNCATE

```sql
SELECT department_id,
    LISTAGG(last_name, '; ') 
    ON OVERFLOW TRUNCATE '...' WITH COUNT
    WITHIN GROUP (ORDER BY hire_date) AS emps
FROM employees
WHERE department_id = 30
GROUP BY department_id;
```

<table>
<thead>
<tr>
<th>DEPARTMENT_ID</th>
<th>EMPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>Clinton; Smith; Jackson; Crain; Clinton; ...</td>
</tr>
</tbody>
</table>

1 row selected

truncated string indicator

count of truncated values
GROUP_CONCAT

• MySQL has a similar function
• Maps directly to LISTAGG
• Implicit CONCAT of values when more than one specified
GROUP_CONCAT

• Similar results to previous example

SQL> SELECT department_id,
            GROUP_CONCAT (last_name
            ORDER BY hire_date
            SEPARATOR '; ') AS emps
FROM module.employees
GROUP BY department_id;

<table>
<thead>
<tr>
<th>DEPARTMENT_ID</th>
<th>EMPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>Clinton; Crain; Jackson; Jackson;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FIRST_VALUE and LAST_VALUE

• Natural extensions from the LISTAGG project
• Uses the ordered set to select the **first** and **last** value
• Includes WITHIN GROUP (ORDER BY ... ) clause
• Not quite the same as MIN and MAX
• Positional values versus computation values
### FIRST_VALUE and LAST_VALUE

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>SALARY_START</th>
<th>SALARY_END</th>
<th>SALARY_AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00369</td>
<td>1982-06-02</td>
<td>NULL</td>
<td>$57,410.00</td>
</tr>
<tr>
<td>00374</td>
<td>1978-10-23</td>
<td>1979-08-19</td>
<td>$43,310.00</td>
</tr>
<tr>
<td>00374</td>
<td>1979-08-19</td>
<td>1980-01-18</td>
<td>$45,427.00</td>
</tr>
<tr>
<td>00374</td>
<td>1980-01-18</td>
<td>1981-01-12</td>
<td>$47,732.00</td>
</tr>
<tr>
<td>00374</td>
<td>1981-01-12</td>
<td>1981-10-15</td>
<td>$47,919.00</td>
</tr>
<tr>
<td>00374</td>
<td>1981-10-15</td>
<td>1982-10-10</td>
<td>$50,424.00</td>
</tr>
<tr>
<td>00374</td>
<td>1982-10-10</td>
<td>NULL</td>
<td>$50,424.00</td>
</tr>
<tr>
<td>00405</td>
<td>1975-07-01</td>
<td>1976-02-26</td>
<td>$27,554.00</td>
</tr>
</tbody>
</table>
RMU Changes
Database Statistics

• Prior releases...
• Database statistics are discarded on RMU/CLOSE
• What if you want to conserve them across database shutdown?
  • Can use RMU /Close /Statistics=EXPORT
    – Creates a .RDS which is a node specific dump of the statistics
  • RMU /Open /Statistics=IMPORT
    – Restores the .RDS to active statistics section
• Also includes /Checkpoint to have the statistics periodically saved
Database Statistics

- What about databases set as OPEN IS AUTOMATIC?
- Added new RMU/Set Statistics command
  - Set and forget
  - Added database attributes to control behavior during OPEN and CLOSE
- Use RMU /Dump /Header to view settings
- Entries are also written to the monitor log file
RMU Set Statistics

• /Export
  Performs an immediate export of the statistics
    – /Export=CLOSE
      On database close the monitor will export the statistics
• /Export=NOCLOSE
  Disables the implicit export
RMU Set Statistics

• /Import=OPEN
  On database open the monitor will import statistics from .RDS
  – /Import=NOOPEN
    Disables the implicit import

• By default the .RDS will be maintained every 30 minutes with a fresh snapshot of the statistics (.RDS is revised)

• /Checkpoint=n
  Can be used to use more or less frequent updates to the .RDS

• /NoCheckpoint
  Disables this checkpointing
RMU Load

• Long standing wish for data type conversion controls in RMU Load

• For example,
  – A CSV file from an external source uses dates in various formats: 17/10/01, 10-Jan-2017
  – A source specifies numbers with 1000 separators 10,000.00, or European style 10.000,00
  – Customer wanted to insert DATE VMS with 7 digit factional precision
  – Treat specific value as a NULL, for example 0 might be considered as NULL
RMU Load

• New syntax for RECORD DEFINITION file (RRD)
• Database administrator can specify SQL routine for transformation of the data
  – STORE USING clause names the routine
  – The assumption is that the routine accepts at least 1 parameter
  – Note: allow routines that default unspecified parameters
• Routine must be a function defined in the database
• SQL or External
• Grant EXECUTE so that RMU Load users can execute routine
RMU Load

- SQL$SAMPLE includes a module of example routines
  - SQL$SAMPLE:CVT_MODULE.SQL

- These can be used directly, or used as the basis for your own

- We will add to these routines
RMU Load – Store Using

```sql
create module EXAMPLE
    function CONVERT_DECIMAL_MARK (in :v varchar(40))
    returns varchar(40)
    comment is 'Preserve digits and sign from the input, '
    /
    'eliminate the digit group separator '
    /
    'and substitute decimal mark';
    return translate (:v, ',', '.');
end module;
```
RMU Load – Store Using Example

DEFINE FIELD EMPLOYEE_ID DATATYPE IS TEXT SIZE IS 5.
DEFINE FIELD SALARY_AMOUNT DATATYPE IS TEXT SIZE IS 30.
DEFINE FIELD SALARY_START DATATYPE IS TEXT SIZE IS 10.
DEFINE FIELD SALARY_END DATATYPE IS TEXT SIZE IS 10.
DEFINE RECORD SALARY_HISTORY.
    EMPLOYEE_ID.
    SALARY_AMOUNT store using CONVERT_DECIMAL_MARK.
    SALARY_START.
    SALARY_END.
END SALARY_HISTORY RECORD.

• One STORE USING per field
• As many for fields that need transformations
Rdb RMU/SHOW STATISTICS Zoom Screen Option

• Available For Logical Area Statistics Screen
  – Displays the top ten or fewer processes attached to an Oracle Rdb database that are accessing the table logical area currently displayed
  – User selects the statistic(s) to be used
    • "ALL" option uses the sum of all the statistics selected
  – The process ID will end with a colon followed by the stream ID assigned by the database
  – Only available for table logical areas
Example: Logical Area Statistics Screen with Zoom Option

<table>
<thead>
<tr>
<th>statistic</th>
<th>rate per second</th>
<th>total</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>max.</td>
<td>curr.</td>
<td>avg.</td>
</tr>
<tr>
<td>A. record marked</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>B. record fetched</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>C. fragmented</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>D. record stored</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>E. fragmented</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>F. pages checked</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>G. saved IO</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>H. discarded</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>I. record erased</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>J. fragmented</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>K. sequential scan</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>L. record fetched</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>M. ALL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Type <return> or <letter> to select logical area statistics, <control-Z> to cancel
Example: Top 10 Processes for ‘Rcord Fetched’

Node: TSTNOD (1/1/16)  
Rate: 3.00 Seconds  
Page: 1 of 1

Oracle Rdb V7.3-300 Perf. Monitor
Logical Area Statistics
DEVICE:[DIRECTORY]MF_PERSONNEL.RDB;1

Table EMPLOYEES in EMPIDS_LOW

...Top Processes Accessing Logical Area EMPLOYEES in EMPIDS_LOW..............
.
.top.........................statistic
.processes.....................record fetched
 20B54CFC:1A  12036
 20B3EB03:1A  10347
 20ADC254:1A  10321
 20A8CE4A:1A  9560
 20B31240:1A  8374
 20B5743A:1A  7312
 20B39E2F:1A  6543
 20A71A11:1A  5478
 20AFC603:1A  4312
 20B46D41:1A  3245
New RMU/RECOVER Optimization

• All transactions must be recovered in the correct original sequence to prevent loss of data and database corruption
  – If the TSN number in the open record of an AIJ file is greater than the highest committed TSN number in the database root file, none of the transactions in the AIJ file will be recovered

• Previously, if the TSN of the AIJ open record was greater, RMU/RECOVER would read through the entire journal file, ignoring all transactions

• Now the recovery operation will be immediately aborted to avoid reading through the entire AIJ file and any additional AIJ files
New RMU/RECOVER Error Messages

• New ERROR Messages if the TSN of the AIJ open record was greater than the highest committed TSN number in the database root file

• Previous Error:
  – %RMU−W−NOTRANAPP, no transactions in this journal were applied

• New Errors output:
  – %RMU−F−BACKUPNOAIJ, After Image Journaling was enabled after the database was backed up or has since been disabled and reinitialized
  – RMU−F−CANTSYNCTSNS, Last committed TSN 96 in the after image journal file exceeds last committed TSN 35 in the database root
  – %RMU−F−TSNNOSYNC, The transactions in this journal file are not consistent with the transactions in this database root file
Example: Journals Enabled AFTER Last Full DB Backup

$ SQL
drop database filename foo.rdb;
exit

$ RMU/RESTORE/NOCDD/NOLOG BAR.RBF
%RMU-I-AIJRSTAVL, 0 after-image journals available for use
%RMU-I-AIJIISOFF, after-image journaling has been disabled
%RMU-W-USERECCOM, Use the RMU Recover command. The journals are not available.
$ RMU/RECOVER/LOG/TRACE FOO.AIJ
%RMU-I-LOGRECDB, recovering database file device:[directory]FOO.RDB;1
%RMU-F-BACKUPNOAIJ, After Image Journaling was enabled after the database was
backed up or has since been disabled and reinitialized
-RMU-F-CANTSYNCTSN, Last committed TSN 96 in the after image journal file
exceeds last committed TSN 35 in the database root
%RMU-F-FTL_RCV, Fatal error for RECOVER operation at 20-SEP-2017 15:02:57.15
Example: No Full Backup After TSN Reset

%! Re-set TSNS
%! $ rmu/repair/initialize=tsns device:[directory]mf_personnel.rdb
%RMU-I-AIJ_ENABLED, This database has after image journaling enabled...
You should create a new journal after this operation completes.
%RMU-I-PULBACREQ, A full backup of this database should be performed after
RMU REPAIR
%! $ Try to apply original .aij; should not succeed
%! $ rmu/recover/log/root=device:[directory]mf_personnel.rdb
device:[directory]pers_aij.aij
%RMU-I-LOGRECDB, recovering database file DISK:[DIRECTORY]MF_PERSONNEL.RDB;2
%RMU-F-TSNNOSYNC, The transactions in this journal file are not consistent
with the transactions in this database root file
-RMU-F-CANTSYNCTSNS, Last committed TSN 448 in the after image journal file
exceeds last committed TSN 0 in the database root
%RMU-F-FTL_RCV, Fatal error for RECOVER operation at 20-SEP-2017 16:08:16.96
New Support for DEFAULT Index NODE SIZE Calculation

• Prior to Oracle Rdb V7.3.1, the default node size was computed as 430 or 860 for longer keys

• Afterwards, maximum node size that could be stored on a page was used as the default

• Some environments would prefer the default NODE SIZE for a sorted index to be smaller than that currently computed by the Rdb.
  – Due to activity (DELETE and UPDATE of key values)
  – Concurrency where the application wants fewer keys to be locked
  – When the page size is very large (e.g. 32 or 63 blocks)
New Support for DEFAULT Index NODE SIZE Calculation

• New RDMS$DEFAULT_INDEX_NODE_SIZE_SMALL logical can be defined to enable an smaller node
  – Similar in size to that of prior versions

• RDMS$SET_FLAGS logical name or SET FLAGS statement can specify 'INDEX_SIZING(SMALL)' to select this algorithm.

• The setting 'INDEX_SIZING(LARGE)' or 'NOINDEX_SIZING' will revert to the other algorithm
New RMU Dump Symbols Command

• Displays or writes to a specified output file the contents of database root file information.
  – Output is similar to that from RMU Dump Header except that the output is in the form of a DCL command procedure that defines global DCL symbols
  – Meant for database administrators who wish to write DCL procedures that react to the current state of the Rdb database
Example: RMU Dump Symbol

CLIENT_FULL_BACKUP_TIMESTAMP

```$ v = 'f$verify(0)
$ set noon
$!
$! Check the last full backup date and see if backup is past due
$!
$ temp_file = "temp" + f$getjpi(0,"PID") + ".tmp;"
$ RMU/DUMP/SYMBOL/EXECUTE/PREFIX=PERS_/OUTPUT=&TEMP_FILE SQL$DATABASE
$ delta_time = f$delta_time (PERS_CLIENT_FULL_BACKUP_TIMESTAMP,"TODAY")
$ days = f$integer(f$element(0," ",delta_time))
$ if days .gt. 7
$ then
$     alert_text = f$fa0("Database PERS not backed up in !SL day!%S", days)
$     write sys$output alert_text
$     ! reply/username=DBADMIN "'alert_text'"
$ endif
$ delete &temp_file
$ exit ! 'f$verify(v)'
```
A few more things...
New logical name: RDMS$BIND_DEADLOCK_WAIT

• Allows sub-second deadlock wait
  – Note: SYSGEN DEADLOCK_WAIT has minimum value of 1 second

• Interface to SYS$SET_PROCESS_PROPERTIESW system service
  – The smallest value is 100000 (which is 10 milliseconds)
  – The largest value is 10000000 (which is 1 second)

• Out of range values are quietly ignored (will use SYSGEN DEADLOCK_WAIT)

• Applied at first Rdb ATTACH (CONNECT, etc) only

• Refer to OpenVMS documentation
TCP/IP Proxy Access

• Now support PROXY access when transport is TCP/IP
• Must have V7.3.2 (or later) on both client and server systems
• Designed to allow easy change from DECnet proxy to TCPIP
• Uses OpenVMS services to access existing proxy database
TCP/IP and Rdb Proxy

• Rdb uses existing proxy database on the remote system
• Uses OpenVMS System services to match incoming credentials with granted proxy access
  – No longer need USER and USING syntax, nor
  – SQL_USERNAME or SQL_PASSWORD (in configuration file)
• Use OpenVMS utility AUTHORIZE to view and create proxy entries
TCP/IP Proxy

• Enabled by default

• Can disable default proxy access using this variable
  SQL_ENABLE_TCPIP_PROXY
  – Can disable on client and/or server

• Should use this variable in the configuration file
  SQL_NETWORK_TRANSPORT_TYPE TCPIP
  – By default Rdb attempts DECnet access first
  – Can improve performance of attach
Questions?
Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.