Rdb Journal
technical corner

Guide to Using SQL: How do I do that? Questions and Answers

A feature of Oracle Rdb

By Ian Smith
Oracle Rdb Relational Technology Group
Oracle Corporation

Guide to Using SQL: How do I do that? Questions and Answers

This article provides the answers to ten frequently asked questions about Rdb and SQL. Most of the solutions require Oracle Rdb V7.1 or later, however, some of the solutions can be applied to older versions of Rdb and this will be noted at the end of each section.

1. How can I find out the name of the database?

The column RDB$FILE_NAME in the system table RDB$DATABASE can be queried to determine the full file specification for the root file (RDB). If you were to use RMU/DUMP to examine the on-disk version of the RDB$DATABASE row you would find that this column is empty. It is computed dynamically so that even after database management commands (RMU/RESTORE, RMU/COPY, etc) the returned name will reflect the correct location.

```
SQL> select rdb$file_name from rdb$database;
   RDB$FILE_NAME
 USERSD_1:[TESTER.DATABASES.V70]MF_PERSONNEL.RDB;1
 >>
 >>
 >>
1 row selected
```

This will work for most current versions of Oracle Rdb. With Rdb V7.1 you can install the RDB$STORAGE_AREAS information table and use that to query the names the other database files.

2. How can I get the current date without time portion for VMS style dates?

CURRENT_DATE returns this information for ANSI style DATE values. Simply use CAST (CURRENT_DATE as DATE VMS) to get the time portion truncated.

Related to this question is how to derive relative dates: YESTERDAY can be calculated as CURRENT_DATE – INTERVAL'1'DAY, and TOMORROW can be calculated as

CURRENT_DATE + INTERVAL'1'DAY. Add a CAST (… AS DATE VMS) to use these values with DATE VMS columns.

OpenVMS does support the special strings 'TODAY', 'TOMORROW', and 'YESTERDAY' that resolve to the expected dates. However, avoid using them in compiled queries because the date is resolved at compile time, and so will be fixed for all future executions of the application.

This syntax has been supported since Rdb V4.1.

3. How can I determine the version of Rdb that is running?

In Rdb 7.1 you can use GET DIAGNOSTICS with the SERVER_IDENTIFICATION option.

```
SQL> declare :si rdb$object_name;
SQL> begin
cont> get diagnostics :si = SERVER_IDENTIFICATION;
cont> end;
SQL> print :si;
 SI
 Oracle Rdb V7.1-04
```

In prior releases there is a simple mapping from the RDB$MAJ_VER field to the Rdb version. Currently the RDB$MIN_VER field is not used so there is no way to fetch the patch release name.

```
SQL> select case rdb$maj_ver
cont>          when 20 then 'V5.1'
cont>          when 21 then 'V6.0'
cont>          when 22 then 'V6.1'
cont>          when 23 then 'V7.0'
cont>          when 24 then 'V7.1'
cont>          when 25 then 'V7.2'
cont>          else NULL
cont>        end
cont> from rdb$database;

 V7.0
1 row selected
```

4. How can I get the name of the node on which my application is running?

The easiest way to do this is by writing your own external function. LIB$GET_HOSTNAME is documented in the OpenVMS HELP file under RTL LIB$. The following procedure definition is used in a simple compound statement.

```
/*
    LIB$GET_HOSTNAME
         The Get Host Node Name routine returns the host node name of the
         local system.

         Format
           LIB$GET_HOSTNAME  hostname [,resultant-length] [,flags]
*/

create procedure LIB$GET_HOSTNAME
    (out :hostname char (80) by descriptor,
     out :resultant_length smallint by reference,
     in :flags integer default 0 by value);
external
    name LIB$GET_HOSTNAME
    location 'SYS$SHARE:LIBRTL.EXE'
    language GENERAL
    parameter style GENERAL;

set flags 'trace';
begin
declare :h char (80);
declare :l int;
call LIB$GET_HOSTNAME (:h, :l);
trace substring (:h from 1 for :l);
end;
```

5: How can I get the name of the application that is running?

Audit triggers often need this information. Starting with Rdb V7.1.0.1 you can use the GET DIAGNOSTICS option IMAGE_NAME to get the file specification of the image. Incidentally, the resulting character string contains the node on which the application is running. If the application is running via remote connection then this will be the image name of the front-end application on the remote node.

```
SQL> declare :app varchar(512);
SQL> begin
cont> get diagnostics :app = IMAGE_NAME;
cont> end;
SQL> print :app;
   APP
 MALIBU::DSA2:[SYS1.SYSCOMMON.][SYSEXE]SQL$71.EXE
 >>
 >>
```

6: How can I easily insert text into a LIST OF BYTE VARYING column?

With Oracle Rdb V7.1 you can use the **insert into cursor … filename …** syntax. This statement is available only in interactive SQL and will read each line from the file and write a new segment into the LIST column.

```
attach 'file mf_personnel';
declare rc insert only table cursor
    for select * from resumes;
declare lc insert only list cursor
    for select resume where current of rc;
open rc;
insert into cursor rc (employee_id) values ('00164');
open lc;
insert into cursor lc filename 'resume.dat' as text;
close lc;
close rc;
```

Here is the file RESUME.DAT

```
Mr. Toliver shows great potential.
He has worked at a variety of companies and now works at Oracle Corporation.
He is a valued employee.
<end>
```

By default the formatting characters CR/LF are inserted at the end of each text line. With Rdb V7.1.2 a new clause AS CHARACTER VARYING can be used to eliminate these extra characters.

7: How can I change the format of data displayed by SELECT in interactive SQL?

There are several commands that can be useful for modifying the output in Interactive SQL, just four will be shown here. The first three examples also work for the most recent versions of Oracle Rdb V7.0.

How can I eliminate the "rows selected" output from a SELECT?

The SET DISPLAY NO ROW COUNTER statement can be used to disable this output for all DML statements in interactive SQL.

```
SQL> select count (employee_id) from job_history;
         274
1 row selected
SQL> set display no row counter;
SQL> select count (employee_id) from job_history;
         274
SQL>
```

For those programmers familiar with Oracle SQL*Plus, you can also use the SET FEEDBACK OFF statement.

How can I suppress the column headings during SELECT?

The SET DISPLAY NO QUERY HEADER statement can be used to disable this output for all DML statements in interactive SQL.

```
SQL> select last_name, first_name from employees where employee_id='00164';
 LAST_NAME        FIRST_NAME
 Toliver          Alvin
1 row selected
SQL> set display no query header;
SQL> select last_name, first_name from employees where employee_id='00164';
 Toliver          Alvin
1 row selected
SQL>
```

For those programmers familiar with Oracle SQL*Plus, you can also use the SET HEADING OFF statement.

How can I have the NULL output of SELECT changed to spaces?

The SET DISPLAY NULL STRING statement can be used to change the string to spaces. Using SET DISPLAY DEFAULT NULL STRING will return the string to the default of 'NULL'.

```
SQL> select job_start, job_end from job_history where employee_id='00164';
 JOB_START     JOB_END
  5-Jul-1980   20-Sep-1981
 21-Sep-1981   NULL
2 rows selected
SQL> set display null string '****';
SQL> select job_start, job_end from job_history where employee_id='00164';
 JOB_START     JOB_END
  5-Jul-1980   20-Sep-1981
 21-Sep-1981   ****
2 rows selected
SQL> set display null string '';
SQL> select job_start, job_end from job_history where employee_id='00164';
 JOB_START     JOB_END
  5-Jul-1980   20-Sep-1981
 21-Sep-1981
2 rows selected
```

For those programmers familiar with Oracle SQL*Plus, you can also use the SET NULL statement.

How can I change the format of the displayed column?

In Rdb V7.1 a new EDIT USING clause was added to allow the programmer to use edit strings directly in the SELECT statement. In prior versions you could do the same thing by defining the column with an EDIT STRING clause, or using a domain that included an EDIT STRING clause.

This new clause is part of the AS clause which allows you to rename the column (i.e. change the query header). We recommend using SQL99 (or SQL92) quoting rules so that mixed case column headings will be used with the AS clause.

This example also makes use of a domain to provide the EDIT STRING. When EDIT USING is presented with a domain name it implicitly uses the EDIT STRING defined for the domain, this is an easy way to provide consistent edit strings in the query output.

```
SQL> set quoting rules 'SQL99';
SQL>
SQL> create domain MONEY integer(2) edit string '$$$,$$$,$$9';
SQL>
SQL> select
cont>     last_name as "Last Name",
cont>     employee_id,
cont>     birthday as "Birthday" edit using 'YYYYBDDBMMM',
cont>     (select salary_amount
cont>      from salary_history sh
cont>      where sh.employee_id = e.employee_id
cont>        and salary_end is null) as "Salary" edit using MONEY
cont> from employees e
cont> where e.employee_id < '00167';
 Last Name        EMPLOYEE_ID   Birthday           Salary
 Toliver          00164         1947 28 Mar       $51,712
 Smith            00165         1954 15 May       $11,676
 Dietrich         00166         1954 20 Mar       $18,497
3 rows selected
```

Note that the EDIT USING clause can be applied to columns or complex value expressions.

9: How can I leave interactive SQL without it prompting me?

The EXIT command will normally check to see if a transaction is active and give you the option of returning to complete the transaction (you can enter any SQL statements one you return). However, using the QUIT statement will automatically ROLLBACK any open transaction and disconnect from active databases.

```
SQL> alter table employees add column new_data int;
SQL> exit
There are uncommitted changes to this database.
Would you like a chance to ROLLBACK these changes (No)? y
SQL> quit
```

If you are in a DEBUG session, then also QUIT from the debugger. The default exit handler will be executed when you EXIT from the debugger and will commit the current transaction.

10: How can I find the rows with duplicate values?

If a **unique** constraint, **primary key** constraint, or a **create unique index** statement fails it may be necessary to query the table to locate these duplicate rows. For example, if we try to create a unique index on the MIDDLE_INITIAL column of the EMPLOYEES table it will result in an error.

```
SQL> create unique index mi_ndx on employees (middle_initial);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDB-E-NO_DUP, index field value already exists;
 duplicates not allowed for MI_NDX
```

Obviously our example uses a poor column to be UNIQUE but it will suffice to illustrate the technique for determining the duplicate rows.

The simplest, and also the most efficient query will be the GROUP BY clause as this performs a single pass over the source table. This is usually the most efficient method, and preferred over a self-join query because it doesn't request any indices for the scan.

```
SQL> select middle_initial, count(*)
cont> from employees
cont> group by middle_initial
cont> having count(*) > 1;
 MIDDLE_INITIAL
 A.                            4
 B.                            4
.
.
.
V.                           5
 NULL                        36
19 rows selected
```

Now we can see that we have many rows with duplicates. In a real example there may only be one or two problem rows. To get more details you can use the preceding query as a derived table to feed an outer query that reports more details.

```
SQL> select employee_id, middle_initial, last_name
cont> from employees,
```

```
cont> (select middle_initial, count(*)
cont>     from employees
cont>     group by middle_initial
cont>     having count(*) > 1) dup (mi, ct)
cont> where dup.mi = employees.middle_initial;
 EMPLOYEE_ID   MIDDLE_INITIAL   LAST_NAME
 00164         A.              Toliver
 00189         A.              Lengyel
 00229         A.              Robinson
 00416         A.              Ames
 00199         B.              Nunez
 00244         B.              Boyd
…etc…
```

Now it is up to the database administrator to correct the problem rows or expand the column set to make the index or constraint unique.

Note that **unique** indices allow only one NULL value to exist for a column in the index, however, an ANSI/ISO SQL99 (or SQL92) **unique** constraint ignores NULL values in columns. This difference might allow you to create a **unique** constraint on a table, but fail to also create a **unique** index.

Oracle Rdb
Guide to Using SQL: How do I do that? Question and Answers
December 2007

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com