COLLABORATE17
TECHNOLOGY AND APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

Use the Oracle Logminer to protect your data and move to the Cloud!

Prepared by: Konstantin Kerekovski, Raymond James Financial

Session ID: 439

BIO

Konstantin Romanov Kerekovski

Work Experience:

- American Express – Oracle DBA 11g, 12c, RAC, GG, DG
- Raymond James Financial – Oracle DBA 11g,12c, Exadata, ODA

Experience with Replication Technologies:

- Goldengate 11g, 12c
- Dataguard 11g, 12c
- Logminer 11g, 12c

What is the Logminer?

- A suite of packages and views
- A tool to analyze your archived logs

Recommended Settings

- Minimum Database Supplemental logging enabled.

- (Optional) Supplemental log groups on tables involved in any Logminer replication.

What is Supplemental Logging?

- Extra redo logging other than what is required to recover database changes via media recovery (RMAN).

Examples:

- Session level information
- Before images of columns

Why is Supplemental Logging important?

- Logminer cannot return valid SQL statements without supplemental logging.
- Logminer will not be usable for any auditing purposes due to missing redo data.

Adding Database Minimum Supplemental Logging

```
SQL> alter database add supplemental log data;

Database altered.
```

Adding Database Minimum Supplemental Logging (cont'd)

```
SQL> select SUPPLEMENTAL_LOG_DATA_MIN from v$database;

SUPPLEME
--------
YES
```

Other Levels of DB Supplemental Logging

```
--Primary Key Columns Only
SQL> alter database add supplemental log data (PRIMARY KEY) columns;

Database altered.


--Unique Key Columns Only
SQL> alter database add supplemental log data (UNIQUE) columns;

Database altered.


--All Columns (NOT RECOMMENDED)
SQL> alter database add supplemental log data (ALL) columns;

Database altered.
```

Database Supplemental Logging considerations

- In 12c, supplemental logging done at CDB$ROOT enables supplemental logging on all PDBs.
- Increases amount of REDO generated by ALL transactions.

Why is table-level Supplemental Logging needed?

- To store before-images of key columns into redo stream.
- To ensure SQL retrieved from SQL_REDO column has the proper predicates in where clause.
- To avoid using ROW_ID's when rerunning SQL_REDO data.

Adding table-level Supplemental Logging

Identification Key Logging

```
ALTER TABLE <SCHEMA>.<TABLE_NAME> ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;

ALTER TABLE <SCHEMA>.<TABLE_NAME> ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;

ALTER TABLE <SCHEMA>.<TABLE_NAME> ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
```

Adding table-level Supplemental Logging

Custom Log Groups

```
ALTER TABLE <SCHEMA>.<TABLE_NAME> ADD SUPPLEMENTAL LOG GROUP <GROUP_NAME>
(<COLUMN_LIST>) ALWAYS;
```

Supplemental Log Test Case

```
SQL> select * from kkerekovski.algebra_class;

        ID FIRST_NAME      LAST_NAME                AGE GRADE
---------- --------------- --------------- ---------- ------
         1 Konstantin      Kerekovski              26 A
         2 John            Doe                     23 A
         3 Jane            Smith                   20 A
```

What happens with no table-level Supplemental Logging?

```
SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2487080

SQL> update kkerekovski.algebra_class set grade='F' where last_name = 'Kerekovski';

1 row updated.

SQL> commit;

Commit complete.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2487104
```

What happens with no table-level Supplemental Logging? (cont'd)

```
SQL> @logmine 2487080 2487104
SQL> set echo on;
SQL> BEGIN
  2  sys.dbms_logmnr.start_logmnr (
  3  startscn => &1,
  4  endscn => &2,
  5  options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
  6  DBMS_LOGMNR.COMMITTED_DATA_ONLY +
  7  DBMS_LOGMNR.CONTINUOUS_MINE +
  8  DBMS_LOGMNR.NO_ROWID_IN_STMT
  9  );
 10  END;
 11  /
old   3: startscn => &1,
new   3: startscn => 2487080,
old   4: endscn => &2,
new   4: endscn => 2487104,

PL/SQL procedure successfully completed.


SQL> select sql_redo from v$logmnr_contents where seg_name = 'ALGEBRA_CLASS';

SQL_REDO
--------------------------------------------------------------------------------
update "KKEREKOVSKI"."ALGEBRA_CLASS" set "GRADE" = 'F' where "GRADE" = 'A';
```

No Supplemental Logging whatsoever

```
SQL> @logmine_rowid 2487080 2487104
SQL> set echo on;
SQL> BEGIN
  2  sys.dbms_logmnr.start_logmnr (
  3  startscn => &1,
  4  endscn => &2,
  5  options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
  6  DBMS_LOGMNR.COMMITTED_DATA_ONLY +
  7  DBMS_LOGMNR.CONTINUOUS_MINE
  8  );
  9  END;
 10  /
old   3: startscn => &1,
new   3: startscn => 2487080,
old   4: endscn => &2,
new   4: endscn => 2487104,

PL/SQL procedure successfully completed.


SQL> col sql_redo format a200
SQL> select sql_redo from v$logmnr_contents where seg_name = 'ALGEBRA_CLASS';

SQL_REDO
--------------------------------------------------------------------------------
update "KKEREKOVSKI"."ALGEBRA_CLASS" set "GRADE" = 'F' where "GRADE" = 'A' and ROWID =
'AAAVlEAAEAAAALPAAA';
```

What happens with all column Supplemental Logging?

```
SQL> alter table kkerekovski.algebra_class add supplemental log data (ALL) columns;

Table altered.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2487782

SQL> update kkerekovski.algebra_class set grade='F' where last_name = 'Kerekovski';

1 row updated.

SQL> commit;

Commit complete.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2487791
```

What happens with all column Supplemental Logging? (cont'd)

```
SQL> @logmine 2487782 2487791
SQL> set echo on;
SQL> BEGIN
  2  sys.dbms_logmnr.start_logmnr (
  3  startscn => &1,
  4  endscn => &2,
  5  options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
  6  DBMS_LOGMNR.COMMITTED_DATA_ONLY +
  7  DBMS_LOGMNR.CONTINUOUS_MINE +
  8  DBMS_LOGMNR.NO_ROWID_IN_STMT
  9  );
 10  END;
 11  /
old   3: startscn => &1,
new   3: startscn => 2487782,
old   4: endscn => &2,
new   4: endscn => 2487791,

PL/SQL procedure successfully completed.

SQL>
SQL> select sql_redo from v$logmnr_contents where seg_name = 'ALGEBRA_CLASS';

SQL_REDO
--------------------------------------------------------------------------------
update "KKEREKOVSKI"."ALGEBRA_CLASS" set "GRADE" = 'F' where "ID" = '1' and "FIRST_NAME" = 'Konstantin' and
"LAST_NAME" = 'Kerekovski' and "AGE" = '26' and "GRADE" = 'A';
```

What happens with PK Supplemental Logging?

```
SQL> alter table kkerekovski.algebra_class add constraint algebra_class_pk primary key (id);

Table altered.

SQL> alter table kkerekovski.algebra_class add supplemental log data (PRIMARY KEY) columns;

Table altered.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2488411

SQL> update kkerekovski.algebra_class set grade = 'F' where last_name = 'Kerekovski';

1 row updated.

SQL> commit;

Commit complete.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2488418
```

What happens with PK Supplemental Logging? (cont'd)

```
SQL> @logmine 2488411 2488418
SQL> set echo on;
SQL> BEGIN
  2  sys.dbms_logmnr.start_logmnr (
  3  startscn => &1,
  4  endscn => &2,
  5  options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
  6  DBMS_LOGMNR.COMMITTED_DATA_ONLY +
  7  DBMS_LOGMNR.CONTINUOUS_MINE +
  8  DBMS_LOGMNR.NO_ROWID_IN_STMT
  9  );
 10  END;
 11  /
old   3: startscn => &1,
new   3: startscn => 2488411,
old   4: endscn => &2,
new   4: endscn => 2488418,

PL/SQL procedure successfully completed.

SQL>
SQL> select sql_redo from v$logmnr_contents where seg_name = 'ALGEBRA_CLASS';

SQL_REDO
--------------------------------------------------------------------------------
update "KKEREKOVSKI"."ALGEBRA_CLASS" set "GRADE" = 'F' where "ID" = '1' and "GRADE" = 'A';
```

Why ***Primary Key*** Identification Key Supplemental Logging is safer

```
SQL> alter table kkerekovski.algebra_class add supplemental log data (PRIMARY KEY) columns;

Table altered.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2487394

SQL> update kkerekovski.algebra_class set grade='F' where last_name = 'Kerekovski';

1 row updated.

SQL> commit;

Commit complete.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2487402
```

Why ***Primary Key*** Identification Key Supplemental Logging is safer (cont'd)

```
SQL> @logmine 2487394 2487402
SQL> set echo on;
SQL> BEGIN
  2  sys.dbms_logmnr.start_logmnr (
  3  startscn => &1,
  4  endscn => &2,
  5  options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
  6  DBMS_LOGMNR.COMMITTED_DATA_ONLY +
  7  DBMS_LOGMNR.CONTINUOUS_MINE +
  8  DBMS_LOGMNR.NO_ROWID_IN_STMT
  9  );
 10  END;
 11  /
old   3: startscn => &1,
new   3: startscn => 2487394,
old   4: endscn => &2,
new   4: endscn => 2487402,

PL/SQL procedure successfully completed.

SQL> select sql_redo from v$logmnr_contents where seg_name = 'ALGEBRA_CLASS';

SQL_REDO
--------------------------------------------------------------------------------
update "KKEREKOVSKI"."ALGEBRA_CLASS" set "GRADE" = 'F' where "ID" = '1' and "FIRST_NAME" = 'Konstantin' and
"LAST_NAME" = 'Kerekovski' and "AGE" = '26' and "GRADE" = 'A';
```

Database Objects of interest

- DBMS_LOGMNR_D
- DBMS_LOGMNR
- V$LOGMNR_CONTENTS

DBMS_LOGMNR_D

```
SQL> desc dbms_logmnr_d
PROCEDURE BUILD
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 DICTIONARY_FILENAME            VARCHAR2                IN     DEFAULT
 DICTIONARY_LOCATION            VARCHAR2                IN     DEFAULT
 OPTIONS                        NUMBER                  IN     DEFAULT
PROCEDURE SET_TABLESPACE
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 NEW_TABLESPACE                 VARCHAR2                IN
```

- Database Package
- Used to store data dictionary in flat files or the redo logs.

The Data Dictionary

- Maps Object IDs to Object Names, aids in DDL tracking.

- Can be accessed in 3 different ways
  1. From stored version in redo logs
  2. From stored version a flat file
  3. From the database itself.

Why you should use 1 of the 3 data dictionary options

```
SQL> @logmine_nodict 2530866 2530876
SQL> BEGIN
  2  sys.dbms_logmnr.start_logmnr (
  3  startscn => &1,
  4  endscn => &2,
  5  options => DBMS_LOGMNR.COMMITTED_DATA_ONLY +
  6  DBMS_LOGMNR.CONTINUOUS_MINE
  7  );
  8  END;
  9  /
old   3: startscn => &1,
new   3: startscn => 2530866,
old   4: endscn => &2,
new   4: endscn => 2530876,

PL/SQL procedure successfully completed.


SQL> select sql_redo from v$logmnr_contents;

SQL REDO
--------------------------------------------------------------------------------
update "UNKNOWN"."OBJ# 88388" set "COL 5" = HEXTORAW('46') where "COL 5" =
HEXTORAW('41') and ROWID = 'AAAVlEAAEAAAALPAAA';
```

DBMS_LOGMNR

```
SQL> desc dbms_logmnr
PROCEDURE ADD_LOGFILE
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 LOGFILENAME                    VARCHAR2                IN
 OPTIONS                        BINARY_INTEGER          IN     DEFAULT
FUNCTION COLUMN_PRESENT RETURNS BINARY_INTEGER
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 SQL_REDO_UNDO                  NUMBER                  IN     DEFAULT
 COLUMN_NAME                    VARCHAR2                IN     DEFAULT
PROCEDURE END_LOGMNR
FUNCTION MINE_VALUE RETURNS VARCHAR2
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 SQL_REDO_UNDO                  NUMBER                  IN     DEFAULT
 COLUMN_NAME                    VARCHAR2                IN     DEFAULT
PROCEDURE REMOVE_LOGFILE
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 LOGFILENAME                    VARCHAR2                IN
PROCEDURE START_LOGMNR
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 STARTSCN                       NUMBER                  IN     DEFAULT
 ENDSCN                         NUMBER                  IN     DEFAULT
 STARTTIME                      DATE                    IN     DEFAULT
 ENDTIME                        DATE                    IN     DEFAULT
 DICTFILENAME                   VARCHAR2                IN     DEFAULT
 OPTIONS                        BINARY_INTEGER          IN     DEFAULT
```

- Starts and ends logmining sessions.
- Specifies which redo logs are to be analyzed.
- Specifies how to display SQL_REDO and SQL_UNDO via OPTIONS argument in START_LOGMNR

START_LOGMNR Options

- NO_SQL_DELIMITER
- NO_ROW_ID_IN_STMT
- COMMITTED_DATA_ONLY
- CONTINUOUS_MINE

- DDL_DICT_TRACKING
- DICT_FROM_ONLINE_CATALOG
- DICT_FROM_REDO_LOGS

COLUMN_PRESENT and MINE_VALUE

```
SQL> select supplemental_log_data_min from v$database;

SUPPLEME
--------
YES

SQL> SHOW USER;
USER is "KKEREKOVSKI"
SQL> create table test_table (
  2  id number, message varchar2(200), extra_info varchar2(200),
  3  constraint test_table_pl primary key
  4  ( id )
  5  );

Table created.

SQL> alter table test_table add supplemental log data (PRIMARY KEY) columns;

Table altered.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2333522

SQL> exec dbms_lock.sleep(5);

PL/SQL procedure successfully completed.

SQL> insert into test_table (id, message) values (1, 'TEST_MESSAGE');

1 row created.

SQL> commit;

Commit complete.

SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    2333526
```

COLUMN_PRESENT and MINE_VALUE (cont'd)

```
SQL> BEGIN
  2  sys.dbms_logmnr.start_logmnr (
  3  startscn => 2333522,
  4  endscn => 2333526,
  5  options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
  6  DBMS_LOGMNR.COMMITTED_DATA_ONLY +
  7  DBMS_LOGMNR.CONTINUOUS_MINE +
  8  DBMS_LOGMNR.NO_ROWID_IN_STMT
  9  );
 10  END;
 11  /


SQL> set lines 100
SQL> col ID_VAL format a5
SQL> col MESSAGE_VAL format a13
SQL> col EXTRA_INFO_VAL format a5
SQL> col BOGUS_VAL format a5
SQL> select
  2  dbms_logmnr.column_present(REDO_VALUE,'KKEREKOVSKI.TEST_TABLE.ID') as ID_PRESENT,
  3  dbms_logmnr.column_present(REDO_VALUE,'KKEREKOVSKI.TEST_TABLE.MESSAGE') as MESSAGE_PRESENT,
  4  dbms_logmnr.column_present(REDO_VALUE,'KKEREKOVSKI.TEST_TABLE.EXTRA_INFO')as EXTRA_INFO_PRESENT,
  5  dbms_logmnr.column_present(REDO_VALUE,'KKEREKOVSKI.TEST_TABLE.BOGUS') BOGUS_PRESENT,
  6  dbms_logmnr.mine_value(REDO_VALUE,'KKEREKOVSKI.TEST_TABLE.ID') ID_VAL,
  7  dbms_logmnr.mine_value(REDO_VALUE,'KKEREKOVSKI.TEST_TABLE.MESSAGE') MESSAGE_VAL,
  8  dbms_logmnr.mine_value(REDO_VALUE,'KKEREKOVSKI.TEST_TABLE.EXTRA_INFO') EXTRA_INFO_VAL,
  9  dbms_logmnr.mine_value(REDO_VALUE,'KKEREKOVSKI.TEST_TABLE.BOGUS') BOGUS_VAL
 10  from v$logmnr_contents
 11  where seg_name = 'TEST_TABLE'
 12  and seg_owner = 'KKEREKOVSKI';

ID_PRESENT MESSAGE_PRESENT EXTRA_INFO_PRESENT BOGUS_PRESENT ID_VA MESSAGE_VAL   EXTRA BOGUS
---------- --------------- ------------------ ------------- ----- ------------- ----- -----
         1               1                  1             0 1     TEST_MESSAGE


SQL> select sql_redo from v$logmnr_contents where seg_owner = 'KKEREKOVSKI' and seg_name = 'TEST_TABLE';

SQL_REDO
--------------------------------------------------------------------------------
insert into "KKEREKOVSKI"."TEST_TABLE"("ID","MESSAGE","EXTRA_INFO") values ('1','TEST_MESSAGE',NULL);
```

V$LOGMNR_CONTENTS

- View exposing REDO and UNDO mined via DBMS_LOGMNR
- System Privilege Needed (12c)
  - LOGMINING
  - SELECT_CATALOG_ROLE
- System Privilege Needed (11g)
  - SELECT ANY TRANSACTION
  - SELECT_CATALOG_ROLE

What happens when V$LOGMNR_CONTENTS is queried?

- Redo is analyzed according to STARTSCN/ENDSCN or STARTTIME/ENDTIME.
- Logfiles are dynamically added to logmining session if CONTINUOUS_MINE was specified.
- REDO and UNDO data is returned in the form of ROWS.
- Change records returned in ascending SCN order unless otherwise specified in SQL statement.

Additional interesting views

- V$LOGMNR_LOGS - Contains additional information on ARCHIVELOGS.

- V$LOGMNR_PARAMETERS – Information on current logmining session.

A little more on V$LOGMNR_PARAMETERS

```
SQL> desc v$logmnr_parameters;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 REQUIRED_START_SCN                                 NUMBER
 REQUIRED_START_DATE                                DATE
 START_DATE                                         DATE
 END_DATE                                           DATE
 START_SCN                                          NUMBER
 END_SCN                                            NUMBER
 OPTIONS                                            NUMBER

---ALWAYS NULL or 0---
 INFO                                               VARCHAR2(32)
 STATUS                                             NUMBER
```

Auditing Database Activity

Database change causes outage

- Database suddenly becomes unresponsive.
- Database server is unable to create any new processes and needs to be rebooted.
- Everyone is panicking and management is very displeased.

What do we know?

1. A heavily used index (HR.EMP_EMP_ID_PK) was recently altered to PARALLEL 99.
2. The database is auditing changes to indexes.

```
SQL> set lines 200
SQL> select audit_option, success,failure from DBA_STMT_AUDIT_OPTS where AUDIT_OPTION
like '%INDEX';

AUDIT_OPTION                             SUCCESS    FAILURE
---------------------------------------- ---------- ----------
ALTER ANY INDEX                          BY ACCESS  BY ACCESS
DROP ANY INDEX                           BY ACCESS  BY ACCESS
```

3. DBA_AUDIT_TRAIL has no audit records for index!

```
SQL> select * from dba_audit_trail where obj_name = 'EMP_EMP_ID_PK';

no rows selected
```

DBMS_LOGMNR to the rescue!

```
SQL> alter session set nls_date_format='YYYY-MM-DD HH24:MI:SS';

Session altered.

SQL> BEGIN
sys.dbms_logmnr.start_logmnr (
  2
  3  starttime => to_date('2017-02-10 20:05:00'),
  4  endtime => to_date('2017-02-10 20:50:00'),
  5  options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + DBMS_LOGMNR.COMMITTED_DATA_ONLY
+ DBMS_LOGMNR.CONTINUOUS_MINE
  6  );
  7  END;
  8  /

PL/SQL procedure successfully completed.
```

Verifying our suspicions

```
SQL> set lines 200 pages 500
SQL> col username format a12
SQL> col os_username format a12
SQL> col sql_redo format a230
SQL> col sql_undo format a60
SQL> select username,os_username,sql_redo
  2  from v$logmnr_contents
  3  where SEG_NAME = 'EMP_EMP_ID_PK'
  4  and os_username = 'kkerekovski'
  5  and operation = 'DDL';

USERNAME     OS_USERNAME  SQL_REDO
------------ ------------ ------------------------------------------------------------
KKEREKOVSKI  kkerekovski  alter index HR.EMP_EMP_ID_PK PARALLEL 99;


SQL> select count(*) from v$logmnr_contents where os_username = 'kkerekovski' and
operation = 'DELETE' and SEG_NAME = 'AUD$';

  COUNT(*)
----------
         3
```

Recovering from Logical Corruption

Botched Update Script

```
SQL> !cat update_salary.sql
set echo on;
update hr.employees set salary = 100000;
where employee_id =100;
commit;
```

Outcome

```
SQL> @update_salary.sql
SQL> update hr.employees set salary = 100000;

107 rows updated.

SQL> where employee_id =100;
SP2-0734: unknown command beginning "where empl..." - rest of
line ignored.
SQL> commit;

Commit complete.
```

DBMS_LOGMNR to the rescue!

```
SQL> alter session set nls_date_format='YYYY-MM-DD HH24:MI:SS';

Session altered.


SQL> BEGIN
  2  sys.dbms_logmnr.start_logmnr (
  3  starttime => to_date('2017-02-10 22:00:00'),
  4  endtime => to_date('2017-02-10 22:05:00'),
  5  options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
  6  DBMS_LOGMNR.COMMITTED_DATA_ONLY +
  7  DBMS_LOGMNR.CONTINUOUS_MINE
  8  );
  9  END;
 10  /

PL/SQL procedure successfully completed.
```

Retrieving the SQL_UNDO

```
SQL> set trimout on
SQL> set lines 4000 pages 0
SQL> select sql_undo from v$logmnr_contents where username = 'SCOTT' and seg_name = 'EMPLOYEES';


update "HR"."EMPLOYEES" set "SALARY" = '2600' where "EMPLOYEE_ID" = '198' and "SALARY" = '100000';
update "HR"."EMPLOYEES" set "SALARY" = '3400' where "EMPLOYEE_ID" = '186' and "SALARY" = '100000';
...

update "HR"."EMPLOYEES" set "SALARY" = '3100' where "EMPLOYEE_ID" = '196' and "SALARY" = '100000';
update "HR"."EMPLOYEES" set "SALARY" = '3000' where "EMPLOYEE_ID" = '197' and "SALARY" = '100000';


107 rows selected.
```

References

https://github.com/kerekovskik/COLLABORATE_2017

https://github.com/kerekovskik/COLLABORATE_2017/wiki

https://docs.oracle.com/database/121/SUTIL/toc.htm

https://docs.oracle.com/database/121/ARPLS/d_sql.htm

COLLABORATE 17

TECHNOLOGY AND APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

Q&A