60
DBMS_CUBE_ADVISE

DBMS_CUBE_ADVISE contains subprograms for evaluating cube materialized views to support log-based fast refresh and query rewrite.

This chapter contains the following topics:

- DBMS_CUBE_ADVISE Security Model
- Summary of DBMS_CUBE_ADVISE Subprograms

> **See Also:**
>
> *Oracle OLAP User's Guide* for information about cube materialized views

DBMS_CUBE_ADVISE Security Model

The MV_CUBE_ADVICE function requires the ADVISOR privilege.

Summary of DBMS_CUBE_ADVISE Subprograms

This table lists and describes the DBMS_CUBE_ADVISE subprograms.

Table 60-1 Summary of DBMS_CUBE_ADVISE Subprograms

Subprogram	Description
MV_CUBE_ADVICE Function	Evaluates the metadata of a cube materialized view and generates recommendations for constraints, SQL dimension objects, and materialized view logs to support a broad range of query rewrite and fast refresh opportunities.
SET_CNS_EXCEPTION_LOG Procedure	Identifies the name of an exception log used in validated constraints generated by MV_CUBE_ADVICE.
TRACE Procedure	Displays or suppresses diagnostic messages for MV_CUBE_ADVICE.

MV_CUBE_ADVICE Function

This table function evaluates the metadata for a specified cube materialized view. It generates recommendations and returns them as a SQL result set. These SQL statements can be used to create constraints, SQL dimension objects, and materialized view logs that allow the broadest range of query rewrite transformations and log-based fast refresh of the cube materialized view.

Syntax

```
DBMS_CUBE_ADVISE.MV_CUBE_ADVICE (
          owner        IN  VARCHAR2  DEFAULT USER,
```

```
        mvname       IN  VARCHAR2,
        reqtype      IN  VARCHAR2  DEFAULT '0',
        validate     IN  NUMBER    DEFAULT 0)
    RETURN COAD_ADVICE_T  PIPELINED;
```

Parameters

Table 60-2 MV_CUBE_ADVICE Function Parameters

Parameter	Description
`owner`	Owner of the cube materialized view
`mvname`	Name of the cube, such as `UNITS_CUBE`, or the cube materialized view, such as `CB$UNITS_CUBE`
`reqtype`	Type of advice to generate: • `0`: All applicable advice types • `1`: Column `NOT NULL` constraints • `2`: Primary key constraints • `3`: Foreign key constraints • `4`: Relational dimension objects • `5`: Cube materialized view logs with primary key
`validate`	Validation option: • `0`: Validate the constraints • `1`: Do not validate the constraints

Returns

A table of type `COAD_ADVICE_T`, consisting of a set of rows of type `COAD_ADVICE_REC`. Table 60-3 describes the columns.

Table 60-3 MV_CUBE_ADVICE Return Values

Column	Datatype	Description
`OWNER`	`VARCHAR2(30)`	Owner of the dimensional object identified in `APIOBJECT`.
`APIOBJECT`	`VARCHAR2(30)`	Name of a cube enhanced with materialized view capabilities, such as `UNITS_CUBE`.
`SQLOBJOWN`	`VARCHAR2(30)`	Owner of the relational object identified in `SQLOBJECT`.
`SQLOBJECT`	`VARCHAR2(65)`	Name of the master table, such as `UNITS_FACT`, or the cube materialized view, such as `CB$UNITS_CUBE`.
`ADVICETYPE`	`NUMBER(38,0)`	Type of recommendation: • `1`: Create `NOT NULL` constraints on the foreign key columns • `2`: Create primary key constraints on the master table • `3`: Create primary key constraints on the master view • `4`: Create foreign key constraints on the master table • `5`: Create foreign key constraints on the master view • `6`: Create relational dimensions on the master dimension tables • `7`: Create a materialized view log • `8`: Compile the materialized view

Table 60-3 (Cont.) MV_CUBE_ADVICE Return Values

Column	Datatype	Description
DISPOSITION	CLOB	Pre-existing conditions that conflict with the recommendations and should be resolved before SQLTEXT can be executed.
SQLTEXT	CLOB	SQL statement that implements the recommendation.
DROPTEXT	CLOB	SQL statement that reverses SQLTEXT. Pre-existing conditions may prevent these statements from restoring the schema to its previous state.

Usage Notes

This function is available in Analytic Workspace Manager as the Materialized View Advisor, which will generate a SQL script with the recommendations.

You can query the returned rows the same as any other table, as shown in the example.

MV_CUBE_ADVICE generates unique object names each time it is called. You should execute the function once, capture the results, and work with those SQL statements.

Take care when dropping database objects. If a table already has a materialized view log, it will have the same name used in the SQL DROP MATERIALIZED VIEW LOG statement in the DROPTEXT column. You should avoid inadvertently dropping materialized view logs, especially when they may be used for remote data replication.

Examples

The following query displays the SQL statements recommended by MV_CUBE_ADVICE. UNITS_FACT is the master table for UNITS_CUBE, and MV_CUBE_ADVICE generates an ALTER TABLE command to add primary key constraints.

It also generates an ALTER MATERIALIZED VIEW command to compile the CB$UNITS_CUBE cube materialized view.

```
SQL> SELECT apiobject, sqlobject, sqltext
      FROM TABLE(dbms_cube_advise.mv_cube_advice('GLOBAL', 'CB$UNITS_CUBE'));

APIOBJECT    SQLOBJECT       SQLTEXT
------------ --------------- --------------------------------------------
UNITS_CUBE   UNITS_FACT      alter table "GLOBAL"."UNITS_FACT" add constra
                             int "COAD_PK000208" PRIMARY KEY ("CHANNEL_ID"
                             , "ITEM_ID", "SHIP_TO_ID", "MONTH_ID") rely d
                             isable novalidate

UNITS_CUBE   CB$UNITS_CUBE   alter materialized view "GLOBAL"."CB$UNITS_CU
                             BE" compile
```

SET_CNS_EXCEPTION_LOG Procedure

This procedure identifies the name of an exception log used in validated constraints generated by `MV_CUBE_ADVICE`.

Syntax

```
DBMS_CUBE_ADVISE.SET_CNS_EXCEPTION_LOG (
       exceptlogtab     IN   VARCHAR2  DEFAULT user.EXCEPTIONS);
```

Parameters

Table 60-4 SET_CNS_EXCEPTION_LOG Procedure Parameters

Parameter	Description
`exceptlogtab`	The name of an existing exception log.

Usage Notes

To create an exception log, use the `utlexcpt.sql` or the `utlexpt1.sql` script before executing `SET_CNS_EXCEPTION_LOG`.

The `validate` parameter of `MV_CUBE_ADVICE` must be set to `1`.

Examples

The `utlexcpt.sql` script creates a table named `EXCEPTIONS`, and the `SET_CNS_EXCEPTION_LOG` procedure identifies it as the exception log for `MV_CUBE_ADVICE`. The `ALTER TABLE` statement now includes the clause `VALIDATE EXCEPTIONS INTO "GLOBAL"."EXCEPTIONS"`.

```
SQL> @utlexcpt
Table created.

SQL> EXECUTE dbms_cube_advise.set_cns_exception_log;
PL/SQL procedure successfully completed.

SQL> SELECT apiobject, sqlobject, advicetype type, sqltext
     FROM TABLE(
     dbms_cube_advise.mv_cube_advice('GLOBAL', 'CB$UNITS_CUBE', '2', 1));

APIOBJECT    SQLOBJECT       TYPE SQLTEXT
------------ --------------- ---- ---------------------------------------------
UNITS_CUBE   UNITS_FACT         2 alter table "GLOBAL"."UNITS_FACT" add constrai
                                  nt "COAD_PK000219" PRIMARY KEY ("CHANNEL_ID",
                                  "ITEM_ID", "SHIP_TO_ID", "MONTH_ID") norely en
                                  able validate exceptions into "GLOBAL"."EXCEPT
                                  IONS"

UNITS_CUBE   CB$UNITS_CUBE      8 alter materialized view "GLOBAL"."CB$UNITS_CUB
                                  E" compile
```

TRACE Procedure

This procedure turns on and off diagnostic messages to server output for the `MV_CUBE_ADVICE` function.

Syntax

```
DBMS_CUBE_ADVISE.TRACE (
     diaglevel       IN  BINARY_INTEGER DEFAULT 0);
```

Parameters

Table 60-5 TRACE Procedure Parameters

Parameter	Description
`diaglevel`	`0` to turn tracing off, or `1` to turn tracing on.

Examples

The following example directs the diagnostic messages to server output. The SQL*Plus `SERVEROUTPUT` setting displays the messages.

```
SQL> SET SERVEROUT ON FORMAT WRAPPED
SQL> EXECUTE dbms_cube_advise.trace(1);
DBMS_COAD_DIAG: Changing diagLevel from [0] to [1]

PL/SQL procedure successfully completed.

SQL> SELECT sqlobject, sqltext, droptext
     FROM TABLE(
     dbms_cube_advise.mv_cube_advice('GLOBAL', 'CB$UNITS_CUBE'))
     WHERE apiobject='UNITS_CUBE';

SQLOBJECT       SQLTEXT                                DROPTEXT
--------------- -------------------------------------- ----------------------------------------
UNITS_FACT      alter table "GLOBAL"."UNITS_FACT" add co alter table "GLOBAL"."UNITS_FACT" drop c
                nstraint "COAD_PK000222" PRIMARY KEY ("C onstraint "COAD_PK000222" cascade
                HANNEL_ID", "ITEM_ID", "SHIP_TO_ID", "MO
                NTH_ID") rely disable novalidate

CB$UNITS_CUBE   alter materialized view "GLOBAL"."CB$UNI alter materialized view "GLOBAL"."CB$UNI
                TS_CUBE" compile                        TS_CUBE" compile


20070706 07:25:27.462780000 DBMS_COAD_DIAG NOTE: Parameter mvOwner  : GLOBAL
20070706 07:25:27.462922000 DBMS_COAD_DIAG NOTE: Parameter mvName   : CB$UNITS_CUBE
20070706 07:25:27.462967000 DBMS_COAD_DIAG NOTE: Parameter factTab  : .
20070706 07:25:27.463011000 DBMS_COAD_DIAG NOTE: Parameter cubeName : UNITS_CUBE
20070706 07:25:27.463053000 DBMS_COAD_DIAG NOTE: Parameter cnsState : rely disable novalidate
20070706 07:25:27.463094000 DBMS_COAD_DIAG NOTE: Parameter NNState  : disable novalidate
20070706 07:25:27.462368000 DBMS_COAD_DIAG NOTE: Begin NN:
20070706 07:25:27.833530000 DBMS_COAD_DIAG NOTE: End   NN:
20070706 07:25:27.833620000 DBMS_COAD_DIAG NOTE: Begin PK:
20070706 07:25:28.853418000 DBMS_COAD_DIAG NOTE: End   PK:
20070706 07:25:28.853550000 DBMS_COAD_DIAG NOTE: Begin FK:
20070706 07:25:28.853282000 DBMS_COAD_DIAG NOTE: End   FK:
20070706 07:25:28.853359000 DBMS_COAD_DIAG NOTE: Begin RD:
20070706 07:25:29.660471000 DBMS_COAD_DIAG NOTE: End   RD:
```

```
20070706 07:25:29.661363000 DBMS_COAD_DIAG NOTE: Begin CM:
20070706 07:25:29.665106000 DBMS_COAD_DIAG NOTE: End   CM:

SQL> EXECUTE dbms_cube_advise.trace(0);
DBMS_COAD_DIAG: Changing diagLevel from [1] to [0]

PL/SQL procedure successfully completed.
```