55
DBMS_CQ_NOTIFICATION

The DBMS_CQ_NOTIFICATION package is part of the database change notification feature that provides the functionality to create registration on queries designated by a client application and so to receive notifications in response to DML or DDL changes on the objects associated with the queries. The notifications are published by the database when the DML or DDL transaction commits.

> **See Also:**
>
> *Oracle Database Development Guide* regarding implementing database change notification.

This chapter contains the following topics:

DBMS_CQ_NOTIFICATION Overview

The DBMS_CQ_NOTIFICATION package provides PL/SQL based registration interfaces. A client can use this interface to create registrations on queries based on objects of interest and specify a PL/SQL callback handler to receive notifications. In case of object level registration, when a transaction changes any of the objects associated with the registered queries and |COMMIT|s, the PL/SQL callback, specified during registration for those objects, is invoked. The application can define client-specific processing inside the implementation of its PL/SQL callback handler.

The interface lets you define a registration block (using a mechanism similar to a BEGIN-END block). The recipient of notifications namely the name of the PL/SQL callback handler and a few other registration properties like time-outs can be specified during the BEGIN phase. Any queries executed subsequently (inside the registration block) are considered "interesting queries" and objects referenced by those queries during query execution are registered. The registration is completed by ENDing the registration block.The registration block lets you create new registrations or add objects to existing registrations.

When a registration is created through the PL/SQL interface, a unique registration ID is assigned to the registration by the RDBMS. The client application can use the registration ID to keep track of registrations created by it. When a notification is published by the RDBMS, the registration ID will be part of the notification.

Typical Applications

This functionality is useful for example to applications that cache query result sets on mostly read-only objects in the mid-tier to avoid network round trips to the database. Such an application can create a registration on the queries it is interested in caching. On changes to objects referenced inside those queries, the database publishes a notification when the underlying transaction commits. In response to the notification, the mid-tier application can refresh its cache by re-executing the query/queries.

DBMS_CQ_NOTIFICATION Security Model

The DBMS_CQ_NOTIFICATION package requires that the user have the CHANGE NOTIFICATION system privilege in order to receive notifications, and be granted EXECUTE privilege on the DBMS_CQ_NOTIFICATION package.

In addition the user is required to have SELECT or READ privileges on all objects to be registered. Note that if the SELECT or READ privilege on an object was granted at the time of registration creation but lost subsequently (due to a revoke), then the registration will be purged and a notification to that effect will be published.

DBMS_CQ_NOTIFICATION Constants

DBMS_CQ_NOTIFICATION constants are used as flag parameters either during registration or when received during the notification.

The DBMS_CQ_NOTIFICATION package has sets of constants:

- EVENT_STARTUP, EVENT_SHUTDOWN, EVENT_SHUTDOWN_ANY, EVENT_DEREG describe the type of the notification published by the database.
- INSERTOP, DELETEOP, UPDATEOP, ALTEROP, DROPOP and UNKNOWNOP describe the type of operation on a table (during a notification published by the database).
- QOS_RELIABLE, QOS_DEREG_NFY, QOS_ROWIDs describe registration Quality of Service properties that the client requires. These are specified during registration.

The constants are shown in the following table.

Table 55-1 DBMS_CQ_NOTIFICATION Constants

Name	Type	Value	Description
ALL_OPERATIONS	BINARY_INTEGER	0	Interested in being notified on all operations, specified as a parameter during registration
ALL_ROWS	BINARY_INTEGER	1	All rows within the table may have been potentially modified
EVENT_STARTUP	BINARY_INTEGER	1	Instance startup notification
EVENT_SHUTDOWN	BINARY_INTEGER	2	Instance shutdown notification
EVENT_SHUTDOWN_ANY	BINARY_INTEGER	3	Any instance shutdown when running Oracle Real Application Clusters (Oracle RAC)
EVENT_DEREG	BINARY_INTEGER	5	Registration has been removed
EVENT_OBJCHANGE	BINARY_INTEGER	6	Notification for object change

Table 55-1 (Cont.) DBMS_CQ_NOTIFICATION Constants

Name	Type	Value	Description
EVENT_QUERYCHANGE	BINARY_INTEGER	7	Notification for query result set change
INSERTOP	BINARY_INTEGER	2	Insert operation
UPDATEOP	BINARY_INTEGER	4	Update operation
DELETEOP	BINARY_INTEGER	8	Delete operation
ALTEROP	BINARY_INTEGER	16	Table altered
DROPOP	BINARY_INTEGER	32	Table dropped
UNKNOWNOP	BINARY_INTEGER	64	Unknown operation
QOS_RELIABLE	BINARY_INTEGER	1	Reliable or persistent notification. Also implies that the notifications will be inserted into the persistent storage atomically with the committing transaction that results in an object change.
QOS_DEREG_NFY	BINARY_INTEGER	2	Purge registration on first notification
QOS_ROWIDS	BINARY_INTEGER	4	Require rowids of modified rows
QOS_QUERY	BINARY_INTEGER	8	Register at query granularity
QOS_BEST_EFFORT	BINARY_INTEGER	16	Best effort evaluation
NTFN_GROUPING_CLASS_TIME	BINARY_INTEGER	1	Group notifications by time
NTFN_GROUPING_TYPE_SUMMARY	BINARY_INTEGER	1	Summary grouping of notifications
NTFN_GROUPING_TYPE_LAST	BINARY_INTEGER	2	Last notification in the group
NTFN_GROUPING_FOREVER	BINARY_INTEGER	-1	Repeat notifications forever

DBMS_CQ_NOTIFICATION Operational Notes

The following are DBMS_CQ_NOTIFICATION operational notes.

Object Level Registration

- The notifications are published by the database when a transaction changes the registered objects and COMMITs.
- All objects referenced in the queries executed inside the registration block starting from the previous NEW_REG_START or ENABLE_REG to REG_END are considered interesting objects and added to the registration.

Query Result Change Registration

- The notifications are published by the database when a transaction changes the result set of the registered query and COMMITs.

Troubleshooting

If you have created a registration and seem to not receive notifications when the underlying tables are changed, please check the following.

- Is the job_queue_processes parameter set to a nonzero value? This parameter needs to be configured to a nonzero value in order to receive PL/SQL notifications through the handler.
- Are the registrations being created as a non-SYS user?
- If you are attempting DML changes on the registered object, are you COMMITing the transaction? Please note that the notifications are transactional and will be generated when the transaction COMMITs.
- It maybe possible that there are run-time errors during the execution of the PL/SQL callback due to implementation errors. If so, they would be logged to the trace file of the JOBQ process that attempts to execute the procedure. The trace file would be usually named <ORACLE_SID>_j*_<PID>.trc. '

  For example, if the ORACLE_SID is 'dbs1' and the process is 12483, the trace file might be named 'dbs1_j000_12483.trc.

  Suppose a registration is created with 'chnf_callback as the notification handler and with registration_id 100. Let us suppose the user forgets to define the chnf_callback procedure. Then the JOBQ trace file might contain a message of the following form.

```
Runtime error during execution of PL/SQL cbk chnf_callback for reg CHNF100
   Error in PLSQL notification of msgid:
   Queue :
   Consumer Name :
   PLSQL function :chnf_callback
   Exception Occured, Error msg:
   ORA-00604: error occurred at recursive SQL level 2
   ORA-06550: line 1, column 7:
   PLS-00201: identifier 'CHNF_CALLBACK' must be declared
   ORA-06550: line 1, column 7:
   PL/SQL: Statement ignored
```

> **See Also:**
>
> For more information about troubleshooting Database Change Notification, see *Oracle Database Development Guide*.

DBMS_CQ_NOTIFICATION Examples

Suppose that a mid-tier application has a lot of queries on the HR.EMPLOYEES table. If the EMPLOYEES table is infrequently updated, it can obtain better performance by caching rows from the table because that would avoid a round-trip to the backend database server and server side execution latency. Let us assume that the application has implemented a mid-tier HTTP listener that listens for notifications and updates the mid-tier cache in response to a notification.

Object Change Registration Example

The DBMS_CQ_NOTIFICATION package can be utilized in this scenario to send notifications about changes to the table by means of the following steps:

1. Implement a mid-tier listener component of the cache management system (for example, using HTTP) that listens to notification messages sent from the database and refreshes the mid-tier cache in response to the notification.
2. Create a server side stored procedure to process notifications

```
CONNECT system;
Enter password: password
GRANT CHANGE NOTIFICATION TO hr;
GRANT EXECUTE ON DBMS_CQ_NOTIFICATION TO hr;

Rem Enable job queue processes to receive notifications.
ALTER SYSTEM SET "job_queue_processes"=2;

CONNECT hr;
Enter password: password
Rem Create a table to record notification events
CREATE TABLE nfevents(regid number, event_type number);

Rem create a table to record changes to registered tables
CREATE TABLE nftablechanges(regid number, table_name varchar2(100),
                            table_operation number);

Rem create a table to record rowids of changed rows.
CREATE TABLE nfrowchanges(regid number, table_name varchar2(100),
                          row_id varchar2(30));

Rem Create a PL/SQL callback handler to process notifications.
CREATE OR REPLACE PROCEDURE chnf_callback(ntfnds IN SYS.CHNF$_DESC) IS
   regid           NUMBER;
   tbname          VARCHAR2(60);
   event_type      NUMBER;
   numtables       NUMBER;
   operation_type  NUMBER;
   numrows         NUMBER;
   row_id          VARCHAR2(20);
 BEGIN
     regid      := ntfnds.registration_id;
     numtables  := ntfnds.numtables;
     event_type := ntfnds.event_type;

  INSERT INTO nfevents VALUES(regid, event_type);
  IF (event_type = DBMS_CQ_NOTIFICATION.EVENT_OBJCHANGE) THEN
    FOR i IN 1..numtables LOOP
      tbname          := ntfnds.table_desc_array(i).table_name;
      operation_type  := ntfnds.table_desc_array(I). Opflags;
      INSERT INTO nftablechanges VALUES(regid, tbname, operation_type);
      /* Send the table name and operation_type to client side listener using
UTL_HTTP */
      /* If interested in the rowids, obtain them as follows */
      IF (bitand(operation_type, DBMS_CQ_NOTIFICATION.ALL_ROWS) = 0) THEN
        numrows := ntfnds.table_desc_array(i).numrows;
      ELSE
        numrows :=0;   /* ROWID INFO NOT AVAILABLE */
      END IF;

      /* The body of the loop is not executed when numrows is ZERO */
      FOR j IN 1..numrows LOOP
          Row_id := ntfnds.table_desc_array(i).row_desc_array(j).row_id;
          INSERT INTO nfrowchanges VALUES(regid, tbname, Row_id);
         /* optionally Send out row_ids to client side listener using UTL_HTTP; */
      END LOOP;

    END LOOP;
  END IF;
  COMMIT;
END;
/
```

In Step 2 we can send as much information about the invalidation as the mid-tier application needs based on the information obtained from the notification descriptor.

Notes

a. In the above example, a registration was created on the EMPLOYEES table with 'chnf_callback' as the PL/SQL handler for notifications. During registration, the client specified reliable notifications (QOS_RELIABLE) and rowid notifications (QOS_ROWIDS)

b. The handler accesses the table descriptor array from the notification descriptor only if the notification type is of EVENT_OBJCHANGE. In all other cases (e.g EVENT_DEREG, EVENT_SHUTDOWN), the table descriptor array should not be accessed.

c. The handler accesses the row descriptor array from the table notification descriptor only if the ALL_ROWS bit is not set in the table operation flag. If the ALL_ROWS bit is set in the table operation flag, then it means that all rows within the table may have been potentially modified. In addition to operations like TRUNCATE that affect all rows in the tables, this bit may also be set if individual rowids have been rolled up into a FULL table invalidation.

   This can occur if too many rows were modified on a given table in a single transaction (more than 80) or the total shared memory consumption due to rowids on the RDBMS is determined too large (exceeds 1% of the dynamic shared pool size). In this case, the recipient must conservatively assume that the entire table has been invalidated and the callback/application must be able to handle this condition.

   Also note that the implementation of the user defined callback is up to the developer. In the above example, the callback was used to record event details into database tables. The application can additionally send the notification details to a mid-tier HTTP listener of its cache management system (as in the example) using UTL_HTTP. The listener could then refresh its cache by querying from the back-end database.

3. Create a registrations on the tables that we wish to be notified about. We pass in the previously defined procedure name (chnf_callback) as the name of the server side PL/SQL procedure to be executed when a notification is generated.

```
Rem Create a REGISTRATION on the EMPLOYEES TABLE
DECLARE
  REGDS      SYS.CHNF$_REG_INFO;
  regid      NUMBER;
  mgr_id     NUMBER;
  dept_id    NUMBER;
  qosflags   NUMBER;
BEGIN
 qosflags := DBMS_CQ_NOTIFICATION.QOS_RELIABLE +
         DBMS_CQ_NOTIFICATION.QOS_ROWIDS;
REGDS := SYS.CHNF$_REG_INFO ('chnf_callback', qosflags, 0,0,0);
regid := DBMS_CQ_NOTIFICATION.NEW_REG_START (REGDS);
SELECT manager_id INTO mgr_id FROM EMPLOYEES WHERE employee_id = 200;
DBMS_CQ_NOTIFICATION.REG_END;
END;
/
```

Once the registration is created in Step 3 above, the server side PL/SQL procedure defined in Step 2 is executed in response to any COMMITted changes to the HR.EMPLOYEES table. As an example, let us assume that the following update is performed on the employees table.

```
UPDATE employees SET salary=salary*1.05 WHERE employee_id=203;COMMIT;
```

Once the notification is processed, you will find rows which might look like the following in the `nfevents`, `nftablechanges` and `nfrowchanges` tables.

```
SQL> SELECT * FROM nfevents;

     REGID     EVENT_TYPE
--------------------------
     20045          6


SQL> SELECT * FROM nftablechanges;

  REGID      TABLE_NAME         TABLE_OPERATION
-----------------------------------------
  20045      HR.EMPLOYEES            4


SQL> select * from nfrowchanges;

    REGID      TABLE_NAME          ROW_ID
-----------------------------------------------------
    20045     HR.EMPLOYEES    AAAKB/AABAAAJ8zAAF
```

Query Result Change Registration Example

1. Creating a Callback

```
CONNECT system;
Enter password: password
GRANT CHANGE NOTIFICATION TO hr;
GRANT EXECUTE ON DBMS_CQ_NOTIFICATION TO hr;
CONNECT hr;
Enter password: password
Rem Create a table to record notification events
CREATE TABLE nfevents(regid NUMBER, event_type NUMBER);

Rem Create a table to record notification queries
CREATE TABLE nfqueries (qid NUMBER, qop NUMBER);

Rem Create a table to record changes to registered tables
CREATE TABLE nftablechanges(
       qid                NUMBER,
       table_name         VARCHAR2(100),
       table_operation    NUMBER);

Rem Create a table to record rowids of changed rows.
CREATE TABLE nfrowchanges(
       qid                NUMBER,
       table_name         VARCHAR2(100),
       row_id             VARCHAR2(2000));


CREATE OR REPLACE PROCEDURE chnf_callback
  (ntfnds IN CQ_NOTIFICATION$_DESCRIPTOR)
IS
  regid           NUMBER;
  tbname          VARCHAR2(60);
  event_type      NUMBER;
  numtables       NUMBER;
  operation_type  NUMBER;
  numrows         NUMBER;
  row_id          VARCHAR2(2000);
```

```
  numqueries      NUMBER;
  qid NUMBER;
  qop NUMBER;

BEGIN
  regid := ntfnds.registration_id;
  event_type := ntfnds.event_type;
  INSERT INTO nfevents VALUES(regid, event_type);
  numqueries :=0;
  IF (event_type = DBMS_CQ_NOTIFICATION.EVENT_QUERYCHANGE) THEN
      numqueries := ntfnds.query_desc_array.count;
      FOR i in 1..numqueries LOOP
         qid := ntfnds.QUERY_DESC_ARRAY(i).queryid;
         qop := ntfnds.QUERY_DESC_ARRAY(i).queryop;
         INSERT INTO nfqueries VALUES(qid, qop);
         numtables := 0;
         numtables := ntfnds.QUERY_DESC_ARRAY(i).table_desc_array.count;
         FOR j IN 1..numtables LOOP
           tbname := ntfnds.QUERY_DESC_ARRAY(i).table_desc_array(j).table_name;
           operation_type := ntfnds.QUERY_DESC_ARRAY(i).table_desc_array(j).Opflags;
           INSERT INTO nftablechanges VALUES(qid, tbname, operation_type);
           IF (bitand(operation_type, DBMS_CQ_NOTIFICATION.ALL_ROWS) = 0)
           THEN
             numrows := ntfnds.query_desc_array(i).table_desc_array(j).numrows;
           ELSE
             numrows :=0;   /* ROWID INFO NOT AVAILABLE */
           END IF;

            /* The body of the loop is not executed when numrows is ZERO */
            FOR k IN 1..numrows LOOP
               Row_id :=
ntfnds.query_desc_array(i).table_desc_array(j).row_desc_array(k).row_id;
               INSERT INTO nfrowchanges VALUES(qid, tbname, Row_id);

           END LOOP;  /* loop over rows */
        END LOOP;     /* loop over tables */
      END LOOP;        /* loop over queries */
  END IF;
  COMMIT;
END;
/
```

2. Creates a query registration

```
DECLARE
  reginfo    cq_notification$_reg_info;
  mgr_id     NUMBER;
  dept_id    NUMBER;
  v_cursor   SYS_REFCURSOR;
  regid      NUMBER;
  qosflags   NUMBER;

BEGIN
  /* Register two queries for result-set-change notifications: */

  /* 1. Construct registration information.
        'chnf_callback' is name of notification handler.
        QOS_QUERY specifies result-set-change notifications. */

   qosflags := DBMS_CQ_NOTIFICATION.QOS_QUERY +
               DBMS_CQ_NOTIFICATION.QOS_ROWIDS;
```

```
   reginfo := cq_notification$_reg_info('chnf_callback', qosflags,0, 0, 0);

  /* 2. Create registration */

        regid := DBMS_CQ_NOTIFICATION.NEW_REG_START(reginfo);

        OPEN v_cursor FOR
           SELECT DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID, manager_id
           FROM HR.employees
           WHERE employee_id = 7902;
        CLOSE v_cursor;

       OPEN v_cursor for
        SELECT DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID, department_id
        FROM HR.departments
        WHERE department_name = 'IT';
       CLOSE v_cursor;

  DBMS_CQ_NOTIFICATION.REG_END;
END;
/
```

3. After creating the query registrations, the output from `USER_CQ_NOTIFICATION_QUERIES` would appear as follows.

```
SQL> SELECT queryid, regid, to_char(querytext)
     FROM user_cq_notification_queries;

   QUERYID      REGID
---------- ----------
TO_CHAR(QUERYTEXT)
--------------------------------------------------------------------------------
        22         41
 SELECT HR.DEPARTMENTS.DEPARTMENT_ID  FROM HR.DEPARTMENTS  WHERE HR.DEPARTMENTS.
DEPARTMENT_NAME  = 'IT'

        21         41
 SELECT HR.EMPLOYEES.MANAGER_ID  FROM HR.EMPLOYEES  WHERE HR.EMPLOYEES.EMPLOYEE_
ID  = 7902


Now, let us perform an UPDATE that changes the result of the query with queryid 22
by renaming the department with name 'IT' to FINANCE.

SQL>  update departments set department_name = 'FINANCE' where department_name =
'IT';

1 row updated.

SQL> commit;

Commit complete.

Now we can query the notifications that we recorded in the callback.

SQL> select * from nfevents;

     REGID EVENT_TYPE
---------- ----------
        61          7


Event type 7 corresponds to EVENT_QUERYCHANGE
```

```
SQL> select * from nfqueries;

       QID        QOP
---------- ----------
        42          7

Event type 7 corresponds to EVENT_QUERYCHANGE

SQL> select * from nftablechanges;
SQL> select * from nftablechanges;

     REGID
----------
TABLE_NAME
--------------------------------------------------------------------------------
TABLE_OPERATION
---------------
        42
HR.DEPARTMENTS
              4


TABLE_OPERATION 4 corresponds to UPDATEOP

SQL> select * from nfrowchanges;
     REGID
----------
TABLE_NAME
--------------------------------------------------------------------------------
ROW_ID
--------------------------------------------------------------------------------
        61
HR.DEPARTMENTS
AAANkdAABAAALinAAF
```

DBMS_CQ_NOTIFICATION Data Structures

The `DBMS_CQ_NOTIFICATION` package defines several `OBJECT` types.

OBJECT Types

- CQ_NOTIFICATION$_DESCRIPTOR Object Type
- CQ_NOTIFICATION$_QUERY Object Type
- CQ_NOTIFICATION$_QUERY_ARRAY Object (Array) Type
- CQ_NOTIFICATION$_TABLE Object Type
- CQ_NOTIFICATION$_TABLE_ARRAY Object (Array) Type
- CQ_NOTIFICATION$_ROW Object Type
- CQ_NOTIFICATION$_ROW_ARRAY Object (Array) Type
- CQ_NOTIFICATION$_REG_INFO Object Type

CQ_NOTIFICATION$_DESCRIPTOR Object Type

This is the top level change notification descriptor type. It is a synonym for the `SYS.CHNF$_DESC` type.

Syntax

```
TYPE SYS.CHNF$_DESC IS OBJECT(
   registration_id    NUMBER,
   transaction_id     RAW(8),
   dbname             VARCHAR2(30),
   event_type         NUMBER,
   numtables          NUMBER,
   table_desc_array   CQ_NOTIFICATION$_TABLE_ARRAY,
   query_desc_array   CQ_NOTIFICATION$_QUERY_ARRAY);
```

Attributes

Table 55-2 CQ_NOTIFICATION$_DESCRIPTOR Object Type

Attribute	Description
`registration_id`	Registration ID returned during registration
`transaction_id`	Transaction ID. `transaction_id` of the transaction that made the change. Will be `NULL` unless the event_type is `EVENT_OBJCHANGE` or `EVENT_QUERYCHANGE`.
`dbname`	Name of database
`event_type`	Database event associated with the notification. Can be one of `EVENT_OBJCHANGE` (change to a registered object), `EVENT_STARTUP`, or `EVENT_QUERYCHANGE`, `EVENT_SHUTDOWN` or `EVENT_DEREG` (registration has been removed due to a timeout or other reason)
`numtables`	Number of modified tables. Will be `NULL` unless the `event_type` is `EVENT_OBJCHANGE`.
`table_desc_array`	Array of table descriptors. Will be `NULL` unless the `event_type` is `EVENT_OBJCHANGE`.
`query_desc_array`	Array of queries changed. This will be `NULL` unless event_type is `EVENT_QUERYCHANGE`

CQ_NOTIFICATION$_QUERY Object Type

The object type describes the changes to a query result caused by an event such as a transaction commit.

An array of `CQ_NOTIFICATION$_QUERY` descriptors is embedded inside the top level notification descriptor (`CQ_NOTIFICATION$_DESCRIPTOR`) for events of type `EVENT_QUERYCHANGE`. The array corresponds to the `SET` of queryids which were invalidated as a result of the event.

This is a synonym for the base type `SYS.CHNF$_QDESC`.

Syntax

```
TYPE SYS.CHNF$_QDESC IS OBJECT (
  queryid            NUMBER,
```

```
  queryop            NUMBER,
  table_desc_array   CQ_NOTIFICATION$_TABLE_ARRAY);
```

Attributes

Table 55-3 TYPE SYS.CQ_NOTIFICATION$_QUERY Object Type

Attribute	Description
queryid	QueryId of the changed query
queryop	Operation describing change to the query
table_desc_array	Array of table changes which contributed to the query Result Set change

CQ_NOTIFICATION$_QUERY_ARRAY Object (Array) Type

This type corresponds to an array of `CQ_NOTIFICATION$_QUERY` objects. It is a synonym for the `SYS.CHNF$_QUERY_ARRAY` type.

Syntax

```
TYPE CQ_NOTIFICATION$_TABLE_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_TABLE;
```

CQ_NOTIFICATION$_TABLE Object Type

This descriptor type describes a change to a table and is embedded inside the top level change notification descriptor type for `EVENT_OBJCHANGE` event types. For query result set changes (event type will be set to `EVENT_QUERYCHANGE`), the array of table descriptors is embedded inside each query change descriptor.

Note that this is a synonym for the type `previously named SYS.CHNF$_TDESC`.

Syntax

```
TYPE SYS.CHNF$_TDESC IS OBJECT (
  opflags            NUMBER,
  table_name         VARCHAR2(2*M_IDEN+1),
  numrows            NUMBER,
  row_desc_array     CQ_NOTIFICATION$_ROW_ARRAY)
```

Attributes

Table 55-4 TYPE SYS.CQ_NOTIFICATION$_TABLE Object Type

Attribute	Description
opflags	Table level operation flags. This is a flag field (bit-vector) that describes the operations that occurred on the table. It can be an `OR` of the following bit fields - `INSERTOP`, `UPDATEOP`, `DELETEOP`, `DROPOP`, `ALTEROP`, `ALL_ROWS`. If the `ALL_ROWS` (0x1) bit is set it means that either the entire table is modified (for example, `DELETE * FROM t`) or row level granularity of information is not requested or not available in the notification and the receiver has to conservatively assume that the entire table has been invalidated.
table_name	Name of modified table

Table 55-4 (Cont.) TYPE SYS.CQ_NOTIFICATION$_TABLE Object Type

Attribute	Description
numrows	Number of modified rows within the table. numrows will be NULL and hence should not be accessed if the ALL_ROWS bit is set in the table change descriptor.
row_desc_array	Array of row descriptors. This field will be NULL if the ALL_ROWS bit is set in opflags.

CQ_NOTIFICATION$_TABLE_ARRAY Object (Array) Type

This type corresponds to an array of CQ_NOTIFICATION$_TABLE objects. It is a synonym for the SYS.CHNF$_TDESC_ARRAY type.

Syntax

```
TYPE CQ_NOTIFICATION$_TABLE_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_TABLE;
```

CQ_NOTIFICATION$_ROW Object Type

An array of CQ_NOTIFICATION$_ROW is embedded inside a CQ_NOTIFICATION$_TABLE (table change descriptor), if the QOS_ROWIDS option was chosen at registration and the ALL_ROWS bit is not set in the opflags field of the table change descriptor.

Note that this is a synonym for the type previously named SYS.CHNF$_RDESC.

Syntax

```
TYPE SYS.CHNF$_RDESC IS OBJECT (
  opflags            NUMBER,
  row_id             VARCAHR2 (2000));
```

Attributes

Table 55-5 TYPE SYS.CQ_NOTIFICATION$_ROW Object Type

Attribute	Description
opflags	Row level operation flags. The flag field (bit vector) describes the operations in the row (could be INSERTOP, UPDATEOP or DELETEOP).
row_id	The rowid of the modified row

CQ_NOTIFICATION$_ROW_ARRAY Object (Array) Type

This object type corresponds to an array of CQ_NOTIFICATION$_ROW objects. It is embedded inside the CQ_NOTIFICATION$_TABLE if QOS_ROWIDS was specified during registration and the ALL_ROWS bit is not set in the opflags field of the table change descriptor.

This type is a synonym for the SYS.CHNF$_RDESC_ARRAY type.

Syntax

```
TYPE CQ_NOTIFICATION$_ROW_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_ROW;
```

CQ_NOTIFICATION$_REG_INFO Object Type

The object type describes the attributes associated with creating a new registration. It is a synonym for the type previously named `SYS.CHNF$_REG_INFO`.

Syntax

```
TYPE SYS.CHNF$_REG_INFO IS OBJECT (
  callback                      VARCHAR2(20),
  qosflags                      NUMBER,
  timeout                       NUMBER,
  operations_filter             NUMBER,
  transaction_lag               NUMBER,
  ntfn_grouping_class           NUMBER,
  ntfn_grouping_value           NUMBER,
  ntfn_grouping_type            NUMBER,
  ntfn_grouping_start_time      TIMESTAMP WITH TIME ZONE,
  ntfn_grouping_repeat_count    NUMBER);
```

Attributes

Table 55-6 TYPE CQ_NOTIFICATION$_REG_INFO Object Type

Attribute	Description
`callback`	Name of the server side PL/SQL procedure to be executed on a notification. Prototype is `<call_backname>(ntfnds IN SYS.chnf$_desc)`
`qosflags`	Quality of service flags. Can be set to an `OR` of the following values: • `QOS_RELIABLE` (`0x1`): Notifications are reliable (persistent) and survive instance death. This means that on an instance death in an Oracle RAC cluster, surviving instances will be able to deliver any queued invalidations. Similarly, pending invalidations can be delivered on instance restart, in a single instance configuration. The disadvantage is that there is a CPU cost/ latency involved in inserting the invalidation message to a persistent store. If this parameter is false, then server side CPU and latency are minimized, because invalidations are buffered into an in memory queue but the client could lose invalidation messages on an instance shutdown. • `QOS_DEREG_NFY` (`0x2`): The registration will be expunged on the first notification • `QOS_ROWIDS` (`0x4`): The notification needs to include information about the rowids that were modified • `QOS_QUERY (0x8)`: specifies query result change notification as opposed to object change notification • `QOS_BEST_EFFORT (0x10)` or `QOS_BEST_EFFORT (0x16)`: can register simplified versions of queries and minimizes evaluation with some false positives.

Table 55-6 (Cont.) TYPE CQ_NOTIFICATION$_REG_INFO Object Type

Attribute	Description
timeout	If set to a nonzero value, specifies the time in seconds after which the registration is automatically expunged by the database. If zero / NULL, the registration lives until explicitly deregistered. Note that the timeout option can be combined with the purge on notification (QOS_DEREG_NFY) option as well.
operations_filter	if nonzero, specifies a filter to be selectively notified on certain operations. These flags can be used to filter based on specific operation types: • 0: Notify on all operations (DBMS_CQ_NOTIFICATION.ALL_OPERATIONS) • 0x2: Notify on every INSERT (DBMS_CQ_NOTIFICATION.INSERTOP) • 0x4: Notify on every UPDATE (DBMS_CQ_NOTIFICATION.UPDATEOP) • 0x8: Notify on every DELETE (DBMS_CQ_NOTIFICATION.DELETEOP) A combination of operations can be specified by using a bitwise OR. Caution: This parameter will be honored for object level registrations but ignored for query result change registrations. To implement notification flow control in 11g, the applications can use the "GROUPING notification" option.
transaction_lag	Lag between consecutive notifications in units of transactions. Can be used to specify the number of transactions/database changes, by which the client is willing to lag behind the database. If 0, it means that the client needs to receive an invalidation message as soon as it is generated. **Caution**: This parameter will be honored for object level registrations but ignored for query result change notification registrations.
ntfn_grouping_class	When grouping notifications, the class based on which the group is derived. Currently, the only allowed value is DBMS_CQ_NOTIFICATION.NTFN_GROUPING_CLASS_TIME by which notifications are grouped by time.
ntfn_grouping_value	The grouping value. This describes the time interval that defines the group in seconds. For example, if this were set to 900, it would mean that notifications that were generated in each 15 minute interval would be grouped together.
ntfn_grouping_type	The type of grouping desired. It can be one of two allowed values • DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_SUMMARY - all notifications in the group are summarized into a single notification • DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_LAST - only the last notification in the group is published and the earlier ones discarded
ntfn_grouping_start_time	When to start generating notifications. If specified as NULL, it defaults to the current system generated time.

Table 55-6 (Cont.) TYPE CQ_NOTIFICATION$_REG_INFO Object Type

Attribute	Description
ntfn_grouping_repeat_count	How many times the notification should be repeated. Set this to DBMS_CQ_NOTIFICATION.NTFN_GROUPING_FOREVER to receive notifications for the life time of the registration. Set to a nonzero value if only a certain number of notifications are desired for the life time of the registration.

Usage Notes

- The type declaration incorporates three other alternative constructors. In the first case all other parameters default to their default values.

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
  callback                      VARCHAR2(20),
  qosflags                      NUMBER,
  timeout                       NUMBER);
```

  The second option applies to the type constructor defined in a previous release, and which is retained for backward compatibility:

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
  callback                      VARCHAR2(20),
  qosflags                      NUMBER,
  timeout                       NUMBER,
  operations_filter             NUMBER,
  transaction_lag               NUMBER);
```

  The third definition contains all the members of the type except `transaction_lag` which is being deprecated:

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
  callback                      VARCHAR2(20),
  qosflags                      NUMBER,
  timeout                       NUMBER,
  operations_filter             NUMBER,
  ntfn_grouping_class           NUMBER,
  ntfn_grouping_value           NUMBER,
  ntfn_grouping_type            NUMBER,
  ntfn_grouping_start_time      TIMESTAMP WITH TIME ZONE,
  ntfn_grouping_repeat_count    NUMBER);
```

- In response to a database change, the server side PL/SQL procedure specified by "callback" is executed. The PL/SQL procedure name has to be specified in the format `schema_name.procedure_name`. The procedure must have the following signature:

```
PROCEDURE <procedure_name>(ntfnds IN SYS.chnf$_desc)
```

  `CHNF$_DESC` describes the change notification descriptor.

- The init.ora parameter `job_queue_processes` must be set to a nonzero value to receive PL/SQL notifications, because the specified procedure is executed inside a job queue process when a notification is generated.

Summary of DBMS_CQ_NOTIFICATION Subprograms

This table lists the `DBMS_CQ_NOTIFICATION` subprograms and briefly describes them.

Table 55-7 DBMS_CQ_NOTIFICATION Package Subprograms

Subprogram	Description
CQ_NOTIFICATION_QUERYID Function	Returns the queryid of the most recent query that was attempted to be registered in a registration block
DEREGISTER Procedure	De-subscribes the client with the supplied registration identifier (ID)
ENABLE_REG Procedure	Begins a registration block using an existing registration identifier (ID)
NEW_REG_START Function	Begins a new registration block
REG_END Procedure	Ends the registration boundary
SET_ROWID_THRESHOLD Procedure	Configures the maximum number of rows of a table published in a change notification if the rows of the table are modified in a transaction

CQ_NOTIFICATION_QUERYID Function

This function returns the queryid of the most recent query that was attempted to be registered in a registration block.

Syntax

```
DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID
 RETURN NUMBER;
```

Return Values

Returns the queryid of the most recently registered query.

DEREGISTER Procedure

This procedure describes the client with the specified registration identifier (ID).

Syntax

```
DBMS_CQ_NOTIFICATION.DEREGISTER (
  regid IN NUMBER);
```

Parameters

Table 55-8 DEREGISTER Procedure Parameters

Parameter	Description
`regid`	Client registration ID

Usage Notes

Only the user that created the registration (or the `SYS` user) will be able to desubscribe the registration.

ENABLE_REG Procedure

This procedure adds objects to an existing registration identifier (ID).

It is similar to the interface for creating a new registration, except that it takes an existing `regid` to which to add objects.Subsequent execution of queries causes the objects referenced in the queries to be added to the specified `regid`, and the registration is completed on invoking the REG_END Procedure.

Syntax

```
DBMS_CQ_NOTIFICATION.ENABLE_REG (
  regid IN NUMBER);
```

Parameters

Table 55-9 ENABLE_REG Procedure Parameters

Parameter	Description
`regid`	Client registration ID

Usage Notes

Only the user that created the registration will be able to add further objects to the registration.

NEW_REG_START Function

This procedure begins a new registration block.

Any objects referenced by queries executed within the registration block are considered interesting objects and added to the registration. The registration block ends upon calling the `REG_END` procedure.

Syntax

```
DBMS_CQ_NOTIFICATION.NEW_REG_START (
  regds IN sys.chnf$_reg_info)
 RETURN NUMBER;
```

Parameters

Table 55-10 NEW_REG_START Function Parameters

Parameter	Description
`sys.chnf$_reg_info`	Registration descriptor describing the notification handler and other properties of the registration

Return Values

The procedure returns a registration-id which is a unique integer assigned by the database to this registration. The registration-id will be echoed back in every notification received for this registration.

Usage Notes

- The only operations permitted inside a registration block are queries (the ones the user wishes to register). DML and DDL operations are not permitted.
- The registration block is a session property and implicitly terminates upon exiting the session. While the registration block is a session property, the registration itself is a persistent database entity. Once created, the registration survives until explicitly deregistered by the client application or timed-out or removed by the database for some other reason (such as loss of privileges).
- The user must have the `CHANGE NOTIFICATION` system privilege and `SELECT` or `READ` privileges on any objects to be registered.
- The `SYS` user will not be permitted to create new registrations.
- Nesting of registration block is not permitted.

REG_END Procedure

This procedure marks the end of the registration block. No newly executed queries are tracked.

Syntax

```
DBMS_CQ_NOTIFICATION.REG_END;
```

SET_ROWID_THRESHOLD Procedure

This procedure configures the maximum number of rows of a table published in a change notification if the rows of the table are modified in a transaction.

Syntax

```
DBMS_CQ_NOTIFICATION.SET_ROWID_THRESHOLD (
  tbname     IN  VARCHAR2,
  threshold  IN  NUMBER);
```

Parameters

Table 55-11 SET_ROWID_THRESHOLD Procedure Parameters

Parameter	Description
`tbname`	Table name qualified by the schema name in the form `schemaname.tablename`
`threshold`	Maximum number of modified rows of the table to be published in the change notification

Usage Notes

- The table needs to be registered for change notification either at object change granularity or at query result set granularity.
- The threshold set by means of this subprogram applies to that instance only and does not persist across instance startup/shutdown.