Oracle Corporate Support
Problem Repository

1. Prob# 1011086.6 ORA-4091: MUTATING TABLES
2. Soln# 2059117.6 BULLETIN ON WORKAROUNDS

1. Prob# 1011086.6 ORA-4091: MUTATING TABLES

Problem ID : 1011086.6
Affected Platforms : Generic: not platform specific
Affected Products : Oracle Server - Enterprise Edition V7
Affected Components : RDBMS Generic
Affected Oracle Vsn : Generic

Summary:
ORA-4091: MUTATING TABLES

+=+

Problem Description:
====================

ORA-4091 is a very common error that occurs with triggers if triggers are not managed properly. A full understanding of triggers will help you avoid that error.

ORA-04091, 00000, "table %s.%s is mutating, trigger/function may not see it"
// *Cause: A trigger (or a user defined plsql function that is referenced in
// this statement) attempted to look at (or modify) a table that was
// in the middle of being modified by the statement which fired it.
// *Action: Rewrite the trigger (or function) so it does not read that table.

A mutating table is a table that is currently being modified by an update, delete, or insert statement. You will encounter the ora-4091 error if you have a row trigger that reads or modifies the mutating table. For example, if your trigger contains a select statement or an update statement referencing the table it is triggering off of you will receive the error.

Another way this error can occur is if the trigger has statements to change the primary, foreign or unique key columns of the table the trigger is triggering off of.

IF you must have triggers on tables that have referential constraints, the workaround is to enforce the referential integrity through triggers as well.

Problem Explanation:
====================

The following bulletin discusses concepts of cascade update, delete, insert and how to avoid the mutating table error.

Additional references: Oracle 7 Server Concepts Guide Ch. 15.
Oracle 7 Server Application Developer's Guide 8-9,8-20.

+==+

Diagnostics and References:

* {6396.6,Y,100} ORA-04091: TABLE IS MUTATING, TRIGGER/FUNCTION MAY NOT SEE IT
* {6397.6,Y,100} DO YOU WANT TO HAVE CASCADE UPDATE, DELETE, INSERT

2. Soln# 2059117.6 BULLETIN ON WORKAROUNDS

Solution ID : 2059117.6
For Problem : 1011086.6
Affected Platforms : Generic: not platform specific
Affected Products : Oracle Server - Enterprise Edition V7
Affected Components : RDBMS Generic
Affected Oracle Vsn : Generic

Summary:
BULLETIN ON WORKAROUNDS

+=+

Solution Description:
=====================

Overview

The purpose of this paper is to illustrate to those customers who require one of the following functional capabilities whilst being able to maintain referential integrity among objects:

- o Cascade Update
- o Cascade Delete
- o Cascade Insert

For cascade Update and Insert functions, using stored triggers and procedures will result in ORA-04091 - Table <table_name> is mutating .....

It must be stressed that this solution should ONLY be used to overcome DML restrictions imposed on triggers in order to maintain referential integrity. Whenever possible it is recommended that normal declarative integrity should be used to maintain foreign key integrity. Enforcing this integrity through stored triggers and procedures will have an effect on performance compared with declarative integrity.

For this solution to work correctly there must be no declarative integrity constraints between objects to enforce the foreign key constraint. The basic principal behind this solution is to suppress the validation checks performed as a result of inserting or updating a foreign key in the CHILD table triggered by a cascade Update or Insert. These checks would normally verify the existence of the new foreign key value in the PARENT record (by SELECTING from the parent table). The suppression of this check is only carried out as a direct result of Cascade Update or Delete, as we can be confident that this new value for the foreign key in the CHILD record does exist (i.e.. a result
of it being inserted or updated in the PARENT table). In all other circumstances
No suppression will take place (e.g.. when changing the DEPTNO of an employee of
when inserting a new employee).

The following code illustrates the how this is achieved for the cascade Update scenario, the code can easily be modified to add the other functionality and encapsulate it all within the same package. The EMP and DEPT table have been used, with the column EMP.DEPTNO in the EMP table referencing DEPT.DEPTNO in the DEPT table.

```
/*************************************************************************

CREATE TABLE DEPT
(DEPTNO         NUMBER          NOT NULL,
 DNAME          VARCHAR2(30),
 LOC            VARCHAR2(30))
/
CREATE TABLE EMP
(EMPNO          NUMBER          PRIMARY KEY,
 ENAME          VARCHAR2(30)    NOT NULL,
 SAL            NUMBER,
 JOB            VARCHAR2(30),
 DEPTNO         NUMBER))
/
*************************************************************************/
/*
The mutation_prevention package is the KEY to the whole solution, it contains
only one variable which will indicate whether integrity checks should be
carried
out as a result of an Insert or Update on the foreign key in the EMP table.
*/

CREATE or replace PACKAGE mutation_prevention AS
  fire_trigger varchar2(5) := 'YES';
 END mutation_prevention;
```

```
/

/*
The package manual_cascade is a general purpose package which can also handle
Insert and Delete functionality
*/

CREATE or replace PACKAGE manual_cascade AS
  PROCEDURE cascade_update(old_key IN NUMBER,
                           new_key IN NUMBER,
                           result  OUT BOOLEAN);
 END manual_cascade;
/

/*
The package body contains the procedures which will handle all the cascade
functionality, each will accept 3 parameters :
         1. old_key -- The Old value which is used to reference CHILD records.
         2. new_key -- The New value which is being inserted or substituted.
         3. result  -- Boolean returning the result of the operation.
Procedures contained in this package will be called by the trigger on the
parent
table [call_manual_cascade].
*/

CREATE or replace PACKAGE BODY manual_cascade AS

  PROCEDURE cascade_update(old_key IN NUMBER,
                           new_key IN NUMBER,
                           result  OUT BOOLEAN) IS


    loop_count NUMBER;
    dummy_2    NUMBER;
    dummy_3    NUMBER;
    l_result   BOOLEAN := false;
    l_old_key  NUMBER;
    l_new_key  NUMBER;
--
--  Declare cursor c1 to lock all child records which may be related to the
--  PARENT record.  The NOWAIT is specified such that the procedure does NOT
--  continue to wait if these records are already locked by another user.
--
    CURSOR c1 (l_dept NUMBER)  IS
      SELECT empno, deptno FROM EMP
        WHERE deptno = l_dept
        FOR UPDATE OF DEPTNO NOWAIT;
--
--  Declare exceptions to gracefully handle the CURSOR irregularities.
--
    RESOURCE_BUSY ExCEPTION;
    INVALID_CURSOR ExCEPTION;

    PRAGMA EXCEPTION_INIT(RESOURCE_BUSY, -54);
    PRAGMA EXCEPTION_INIT(INVALID_CURSOR, -1001);

  BEGIN
--
--  Assign input variables to local variables.
--
    l_old_key := old_key;
    l_new_key := new_key;
--
-- Check to see if there are any related child records to be updated.  If NOT
-- then exit the IF TEST, otherwise proceed in locking the rows.
--
    SELECT count(*) INTO loop_count
    FROM   EMP
    WHERE  deptno = l_old_key;

    IF loop_count >= 1 THEN
--
--  At this stage you could define an REPEAT LOOP which will retry this
--  transaction in case of failure to lock all CHILD records successfully
--  [n] time.  This concept is detailed in the PL/SQL manual V2.0 page 5-23
--  in the Error Handling section.
```

```
--
      BEGIN
--
--  Open the cursor and handle any Error Conditions.
--
        OPEN c1 (l_old_key);

        IF NOT  (c1%ISOPEN) THEN
           RAISE INVALID_CURSOR;
        END IF;
--
--  If successful then loop and update each row, one at a time until no more
--  rows.  Handle Exceptions and Close the cursor.
--
        FOR i IN 1..loop_count LOOP
           FETCH c1 INTO dummy_2, dummy_3;

           UPDATE emp SET deptno = l_new_key WHERE CURRENT OF c1;
        END LOOP;

      EXCEPTION
        WHEN INVALID_CURSOR THEN
             result := true;
        WHEN RESOURCE_BUSY THEN
             result := true;
        WHEN OTHERS        THEN
             raise_application_error(-20006,'General Package Error');
             CLOSE c1; ----- ?

      END;
    END IF;
  END cascade_update;
END manual_cascade;
/

/*
This trigger is on the PARENT table DEPT and controls the value of the
global variable [mutation_prevention.fire_trigger] which will be persistent
for the duration of the user session.
*/

CREATE or replace TRIGGER call_manual_cascade
AFTER UPDATE OF deptno ON DEPT
FOR EACH ROW

DECLARE
  l_result           BOOLEAN;
  transaction_failed EXCEPTION;
  debug_var          varchar2(5);
BEGIN
--
--  Set the global control variable indicating to the [TRIGGER EMP_DEPT_CHECK]
--  trigger which performs the foreign key integrity check, that NO check
--  should be performed at this stage.
--
  mutation_prevention.fire_trigger := 'NO';
--
--  This is for debugging only and should be commented out on being satisfied
--  that the global variable is being set correctly.
--
  debug_var:= mutation_prevention.fire_trigger;
  dbms_output.put_line(debug_var);
--
--  Check to see which function is being performed, execute the appropriate
--  procedure in the package and the RESET the value of the global variable
--  to 'YES' to allow subsequent integrity checks to be performed.
--
  IF UPDATING THEN
     manual_cascade.cascade_update(:old.deptno, :new.deptno, l_result);
     mutation_prevention.fire_trigger := 'YES';
--
--  Debugging only, check RESET of global variable.
--
     debug_var:= mutation_prevention.fire_trigger;
     dbms_output.put_line(debug_var);
--
```

```
--  Check the results from the executed procedure .. act accordingly, If this
--  was a BEFORE update Trigger the OLD value for DEPTNO could be rest if
--  required.
--
     IF l_result = TRUE THEN
--         :new.deptno := :old.deptno;
        RAISE transaction_failed;
     END IF;
  END IF;

EXCEPTION
  WHEN transaction_failed THEN
     raise_application_error(-20001,
       'Update Of Department '||to_char(:old.deptno)||' FAILED');
  WHEN Others THEN
     raise_application_error(-20004,'GENERAL ERROR');
END;
/

/*
This trigger performs the integrity checking to validate that the foreign key
constraint is NOT violated when inserting and new employee or updating an
employees department.
*/

CREATE OR replace TRIGGER EMP_DEPT_CHECK
BEFORE UPDATE ON emp
FOR EACH ROW
DECLARE

  l_fire_trigger varchar2(5);
  dummy          number(5);
--
--  Declare a mutating table EXCEPTION, however if this solution works this
--  should never be invoked, but has been included for completeness.
--
  mutating_table EXCEPTION;
  PRAGMA EXCEPTION_INIT (mutating_table, -4091);

BEGIN
--
--  Check the Global variable, and either continue processing or EXIT.
--
    l_fire_trigger := mutation_prevention.fire_trigger;

    IF l_fire_trigger = 'YES' THEN
     BEGIN
        Select 1
        into dummy
        FROM   dept
        WHERE  deptno = :new.deptno;
      IF dummy != 1 THEN
         RAISE NO_DATA_FOUND;
      END IF;
      EXCEPTION
          WHEN NO_DATA_FOUND THEN
             raise_application_error(-20002,
                'Department '||to_char(:new.deptno)||' is NOT VALID');
          WHEN mutating_table then
             raise_application_error(-20003,
                 'Table is MUTATING !!!!!!');
      END;
     END IF;

END;
/
/************************* CONCEPTUAL MODEL ******************************

 -------------    1. Trigger on PARENT for cascade function
 |           |       SET global_variable = 'NO';
 |  PARENT   |          do cascade update;
 |           |      SET global_variable = 'YES';
 -------------     End;
       |                                              -------------------
       |                                              | GLOBAL_VARIABLE |
       |                                              -------------------
```

```
     /|\
 ------------  2. Trigger on CHILD to maintain referential integrity
 |          |       CHECK global_variable;
 |  CHILD   |              IF global_variable = 'YES' THEN
 |          |               check existence of (fk) in PARENT;
 ------------           ELSE
                             null;
                     End;
*****************************************************************************/
```

Solution Explanation:
=====================