

Work Around for Oracle Mutating Tables Error

Lev Moltyaner
Apr 18, 2000

Mutating Table Restriction

- **A mutating table is a table that is currently being modified by an UPDATE, DELETE, or INSERT**

- **In the following example, the INSTRUCTOR table is mutating:**

```
UPDATE INSTRUCTOR  
SET salary = 5500  
WHERE instructor_name = 'STEEL';
```

- **To understand the effect of mutating tables on triggers, we will implement a simple business requirement**
 - Business rule states that an instructor's salary must be less than his or her mentor's salary
 - The business rule will be enforced by a trigger that fires based on insert or update of the INSTRUCTOR table

Mutating Tables: Example

```
CREATE OR REPLACE TRIGGER inst_after_ins_upd
AFTER INSERT OR
    UPDATE OF salary, mentor_id
ON instructor
FOR EACH ROW
DECLARE dummy number;
BEGIN
    IF:new.mentor_id is NOT NULL AND
        (:new.salary>:old.salary OR
        :old.mentor_id<>:new.mentor_id) THEN
        BEGIN
            SELECT      1
            INTO  dummy
            FROM  instructor
            WHERE      instructor_id = :new.mentor_id
            AND        salary > :new.salary;
        EXCEPTION WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000,
                'Instructor salary must be less than mentor salary');
        END;
    END IF;
END;
```

- **During the update, the INSTRUCTOR table is in the mutating state, thus the SELECT on the INSTRUCTOR table in a row-level trigger will result in a mutating table error**
 - ORA-4091: Table oral.instructor is mutating; trigger may not read or modify it

Defining the Problem

- **We need to read the INSTRUCTOR table during the UPDATE to get the mentor's salary for comparison to the instructor's salary**
- **Because the data in the INSTRUCTOR table is being modified during an UPDATE, the table is mutating**
- **Therefore, the data is in an inconsistent state as far as the transaction is concerned**
- **The after row trigger references the INSTRUCTOR table in a query**
- **Oracle detects that a row trigger is accessing inconsistent data and raises the error**
 - Ensures that inconsistent data is not being accessed during the transaction
- **The mutating table problem occurs in row-level triggers**
- **It is not encountered in statement-level triggers**
 - They do not have to access table using :new and :old variables
- **PL/SQL tables can be used to work around the mutating table problem**

Resolving the Problem

- **Solve the mutating table problem by moving the violating code from a row-level trigger to a statement-level trigger**
 - Mutating table problem does not affect statement-level triggers
 - In the example, the violating code was the `SELECT` statement
- **:NEW and :OLD variables are not available in statement triggers**
 - In the row-level trigger, save the required data in persistent variables
 - Persistent variables then accessed from the statement-level trigger
 - Need at least one variable per affected row
- **Because the data must be persistent, the PL/SQL tables must be defined in a package**
- **Once all rows are processed, the PL/SQL tables will contain salaries and mentor_ids of the affected rows**
- **Data can then be accessed in the after statement trigger to perform the required validation**

PL/SQL Tables

- **A PL/SQL table is an array in memory**
- **Modeled like a database table, but has only two columns**
 - First column stores the required information
 - Any scalar data type (`NUMBER`, `VARCHAR2`, `DATE`)
 - Second column is the primary key
 - Used to point to the row that contains the required information
 - Must be `BINARY_INTEGER`
- **PL/SQL tables have been enhanced in PL/SQL Release 2.3**
 - Allows the use of table types defined with `%ROWTYPE` to use more than a single column

Declaring PL/SQL Tables

- **A PL/SQL table is declared in two steps**

- Define the structure of the PL/SQL table
- First part of the definition is the data type
- Use %TYPE to base a PL/SQL table on a database column

```
TYPE pl_sql_table_type
```

```
    IS TABLE OF table_name.column_name%TYPE
```

Define instr_mentor_id_type based on the mentor_id column of
the instructor table

```
TYPE inst_mentor_id_type IS TABLE OF
```

```
    instructor.mentor_id%TYPE INDEX BY BINARY_INTEGER;
```

- Declare a PL/SQL table based on this structure
- Declared like any other variable

```
inst_mentor_id      inst_mentor_id_type;
```

- **A second PL/SQL table, inst_salary_type, will be defined based on the salary column to store salaries**

```
TYPE inst_salary_type IS TABLE OF
```

```
    instructor.salary%TYPE INDEX BY BINARY_INTEGER;
```

```
inst_salary        inst_salary_type;
```

Referencing a PL/SQL Table

- **A table row is referenced using the primary key pointer**

```
plsql_table_name(primary_key_pointer);
```

Reference the 10th row of the salary table

```
inst_salary (10);
```

- **Declare a variable to used as a pointer for the PL/SQL tables**

– Must be of the same data type as defined by the INDEX BY clause

```
inst_pk          BINARY_INTEGER;
```

Assigning Values to a PL/SQL Table

- **A value can be assigned to a specific row in the table**

```
plsql_table_name(primary_key_pointer) := expression;
```

```
inst_salary (inst_pk) := salary;
```

- **The expression must return a compatible data type**

Retrieving Values From a PL/SQL Table

- **Data can be retrieved by assigning it to a variable using the pointer**

– Because this is an array, do not have to use a SELECT to retrieve the data from the PL/SQL table

```
variable := plsql_table_name(primary_key_pointer);
```

```
v_salary := inst_salary (inst_pk);
```

Solving the Mentor's Salary Problem

- **To compare the mentor's salary to the instructor's salary, put `mentor_id` and `salary` into the PL/SQL tables**
- **Use the data in these temporary tables for the comparison**
- **PL/SQL tables can be used in any valid PL/SQL block**
- **To solve the salary problem, use a package**
 - Package used so variable value is maintained across multiple procedures

The Road Map

- **First, define the components (the road map) that will be needed to solve the problem**
- **Package specification**
 - Contains declarations of three procedures that need to be called by triggers
 - First procedure to initialize the pointer to zero
 - Second procedure accepts `mentor_id` and `salary` as parameters and populates the PL/SQL tables with these values
 - Third procedure will do the salary comparison and raise an error if required
- **Package body contains**
 - Definitions for PL/SQL tables and pointer
 - The required procedures
- **Before statement trigger**
 - Calls the procedure to initialize PL/SQL table pointer
 - Fired once before each triggering statement
 - PL/SQL tables are persistent in the scope of the package
 - Must be initialized every time a triggering statement is fired
- **After row trigger**
 - Calls the procedure to save `:NEW` variables in PL/SQL tables

- **After statement trigger**
 - Calls the procedure to perform the required check using PL/SQL tables
 - Table is no longer mutating in AFTER statement trigger

Update Statement Processing

- **Validate the design by reviewing how an UPDATE statement will be processed**
 - First, the before statement trigger is fired and the PL/SQL tables are initialized
 - Next, the first row is updated and the after row trigger is fired
 - The salary and mentor_id for the row being processed are stored in the PL/SQL tables
 - Fired once for every row affected by the UPDATE statement
 - After all rows are updated, the after statement trigger is fired
 - Data corresponding to each affected row is read from PL/SQL tables and used for the validation
 - Design is complete

Package Specification

```
CREATE OR REPLACE PACKAGE inst_pack IS
    PROCEDURE init_inst_plsql_tab_pk;
    PROCEDURE ins_inst_plsql_tab
        (in_mentor_id IN instructor.mentor_id%TYPE
        ,in_salary IN instructor.salary%TYPE);
    PROCEDURE chk_inst_plsql_tab;
END inst_pack;
/
```

Package Body

```
CREATE OR REPLACE PACKAGE BODY inst_pack IS
    TYPE inst_mentor_id_type IS TABLE OF instructor.mentor_id%TYPE
    INDEX BY BINARY_INTEGER;
    inst_mentor_id inst_mentor_id_type;

    TYPE inst_salary_type IS TABLE OF instructor.salary%TYPE
    INDEX BY BINARY_INTEGER;
    inst_salary inst_salary_type;

    inst_pk binary_integer;

    PROCEDURE init_inst_plsql_tab_pk IS
    BEGIN
        inst_pk := 0;
    END;

    PROCEDURE ins_inst_plsql_tab
    (in_mentor_id IN instructor.mentor_id%TYPE
    ,in_salary IN instructor.salary%TYPE) IS
    BEGIN
        inst_pk := inst_pk + 1;
        inst_mentor_id(inst_pk) := in_mentor_id;
        inst_salary(inst_pk) := in_salary;
    END;

    PROCEDURE chk_inst_plsql_tab IS
        dummy number;
    BEGIN
        WHILE inst_pk > 0 LOOP
            BEGIN
                SELECT 1
                INTO      dummy
                FROM      instructor
                WHERE     instructor_id=inst_mentor_id(inst_pk)
                AND       salary > inst_salary(inst_pk);
            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                    RAISE_APPLICATION_ERROR(-20000,
                    'Instructor salary must be less than mentor salary');
            END;
            inst_pk := inst_pk - 1;
        END LOOP;
    END;
END inst_pack;
/
```

Before Statement Trigger

```
CREATE OR REPLACE TRIGGER inst_before_st
BEFORE INSERT OR
      UPDATE OF salary,mentor_id
ON instructor
BEGIN
    inst_pack.init_inst_plsql_tab_pk;
END;
/
```

After Row Trigger

```
CREATE OR REPLACE TRIGGER inst_after_ins_upd
After INSERT OR
      UPDATE OF salary, mentor_id
ON instructor
FOR EACH ROW
BEGIN
    IF :new.mentor_id IS NOT NULL THEN
        BEGIN
            inst_pack.ins_inst_plsql_tab (:new.mentor_id, :new.salary);
        END;
    END IF;
END;
/
```

After Statement Trigger

```
CREATE OR REPLACE TRIGGER inst_after_st
After INSERT OR
      UPDATE OF salary,mentor_id
ON instructor
BEGIN
    inst_pack.chk_inst_plsql_tab;
END;
/
```