PL/SQL

Triggers

Overview

- What are Triggers?
- Procedures Vs Triggers
- Trigger Issues
- When to use Triggers?
- ECA Model
- Syntax
- Row vs Statement Triggers
- Before Vs After Triggers

TRIGGERS

- A database trigger is a stored PL/SQL program unit associated with a specific table, view, schema, or the database.
- ORACLE executes (fires) a database trigger automatically when a given SQL operation (like INSERT, UPDATE or DELETE) affects the table.
- Triggers are action that executes (fire) automatically whenever some specified event occurs.

TRIGGERS

- Can be either:
  - Application trigger: fires when an event occurs with a particular application
  - Database trigger: fires when a data event (DML) or system event (logon or shutdown) occurs on schema or database.

Procedures vs. Triggers

- Procedures and triggers differ in the way that they are invoked:
  - A procedure is explicitly executed by a user, application, or trigger.
  - Triggers are implicitly fired when a triggering event occurs, no matter which user is connected or which application is being used.
- Note that the database stores triggers separately from their associated tables.

Issues

- Triggers are not recommended way to implement integrity constraints.
- Declarative constraints are checked on all relevant updates, whereas triggers are invoked only when the specified event occurs.
- Trigger T1 could cause trigger T2 to fire, which could cause trigger T3 and so on (a trigger chain).
- Trigger T might even cause itself to fire again (recursive).

When to use triggers?

- Some useful purpose of triggers:
- Alerting the user if some exception occurs
- Debugging (e.g., monitoring references to, and/or state changes in, designated variables)
- Auditing (e.g., tracking who performed what updates to which relations when)
- Performance measurement (e.g., timing or tracking specified database events)
- Carrying out compensating actions.
- Implement complex security authorizations
- Enforce complex business rules

ECA model

- Trigger specifies, an event, a condition, and an action:
  - The event is an operation on the database (any update operations).
  - The condition is a boolean expression that has to evaluate to TRUE in order for the action to be executed. (if no condition, then the default is just TRUE)
  - The action is the triggered procedure.
- The event and condition together are called the triggering event
- Triggers are also known as event-condition-action rules [ECA]

ECA model

- Events include INSERT, DELETE, UPDATE, end-of-transaction, reaching specified time-of-day, violating a specified constraint, and so on.
- The action can be performed BEFORE, AFTER, or INSTEAD OF the specified event.
- The action can be performed FOR EACH ROW or FOR EACH STATEMENT.
- A database that has associated triggers is sometimes called an active database.

TRIGGERS - SYNTAX

```
CREATE [OR REPLACE] TRIGGER trigger_name
[BEFORE|AFTER] [ INSERT | UPDATE | DELETE ]
ON table_name
[FOR EACH ROW]
[WHEN condition]
BEGIN
…
…
PLSQL Block
…
…
END;
```

TRIGGERS

- The trigger_name references the name of the trigger.
- BEFORE or AFTER specify when the trigger is fired (before or after the triggering event).
- The triggering_event references a DML statement issued against the table (e.g., INSERT, DELETE, UPDATE).
- The table_name is the name of the table associated with the trigger.
- The clause, FOR EACH ROW, specifies a trigger is a row trigger and fires once for each modified row.
- A WHEN clause specifies the condition for a trigger to be fired.
- Bear in mind that if you drop a table, all the associated triggers for the table are dropped as well.

TYPES OF TRIGGERS

- Row Triggers and Statement Triggers
- BEFORE and AFTER Triggers
- INSTEAD OF Triggers
- Triggers on System Events and User Events

Row Trigger vs Statement Trigger

- A row trigger is fired each time the table is affected by the triggering statement.
- For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement.
- The FOR EACH ROW option determines whether the trigger is a row trigger or a statement trigger.

Row Trigger vs Statement Trigger

- A statement trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even if no rows are affected.
- The absence of the FOR EACH ROW option indicates that the trigger fires only once for each applicable statement, but not separately for each row affected by the statement.

BEFORE vs AFTER triggers

- When defining a trigger, you can specify the trigger timing – whether the trigger action is to be executed before or after the triggering statement.
- BEFORE and AFTER apply to both statement and row triggers.
- BEFORE – For row triggers, the trigger is fired before each affected row is changed.
- AFTER – For row triggers, the trigger is fired after each affected row is changed.

BEFORE vs AFTER triggers

- Restrictions on AFTER Triggers:
  - You cannot specify an AFTER trigger on a view or an object view.
  - You cannot write either the :OLD or the :NEW value.
- Restrictions on BEFORE Triggers:
  - You cannot specify a BEFORE trigger on a view or an object view.
  - You can write to the :NEW value but not to the :OLD value.

Conditional predicates

- More than one type of DML operation can fire a trigger:
- Ex: ON INSERT OR DELETE OR UPDATE OF salary
- The trigger body can use the conditional predicates INSERTING, DELETING, and UPDATING to check which type of statement fire the trigger.

Ex:

```
IF INSERTING THEN …. …. END IF;
IF DELETING THEN …. …. END IF;
IF UPDATING ('SAL') THEN … … END IF;
```

Column values in row triggers

- The PL/SQL code and SQL statements have access to the old and new column values of the current row affected by the triggering statement.
- The new column values are referenced using the new qualifier while the old column values are referenced using the old qualifier before the column name.

Example:

```
IF :new.sal > 50000 ....
IF :new.sal < :old.sal ....
```

Example - Statement trigger

```
CREATE OR REPLACE TRIGGER mytrig1 BEFORE
    DELETE OR INSERT OR UPDATE ON dept
BEGIN
IF (TO_CHAR(SYSDATE, 'day') IN ('sat', 'sun')) OR
    (TO_CHAR(SYSDATE,'hh:mi') NOT BETWEEN
    '08:30' AND '18:30') THEN
    RAISE_APPLICATION_ERROR(-20500, 'table is
    secured');
END IF;
END;
```

- This example shows a trigger that limits the DML actions to the dept table to weekdays from 8.30am to 6.30pm. If a user tries to insert/update/delete a row in the DEPT table, a warning message will be prompted.

Example - ROW Trigger

```
CREATE OR REPLACE TRIGGER mytrig2
AFTER DELETE OR INSERT OR UPDATE ON employee
FOR EACH ROW
BEGIN
IF DELETING THEN
INSERT INTO xemployee (emp_ssn, emp_last_name,emp_first_name, deldate)
VALUES (:old.emp_ssn, :old.emp_last_name,:old.emp_first_name, sysdate);
ELSIF INSERTING THEN
 INSERT INTO nemployee (emp_ssn, emp_last_name,emp_first_name,
      adddate)
VALUES (:new.emp_ssn, :new.emp_last_name,:new.emp_first_name, sysdate);
 ELSIF UPDATING('emp_salary') THEN
INSERT INTO cemployee (emp_ssn, oldsalary, newsalary, up_date)
VALUES (:old.emp_ssn,:old.emp_salary, :new.emp_salary, sysdate);
ELSE
INSERT INTO uemployee (emp_ssn, emp_address, up_date)
VALUES (:old.emp_ssn, :new.emp_address, sysdate);
END IF;
END;
```

Example

- The previous trigger is used to keep track of all the transactions performed on the employee table. If any employee is deleted, a new row containing the details of this employee is stored in a table called xemployee. Similarly, if a new employee is inserted, a new row is created in another table called nemployee, and so on.
- Note that we can specify the old and new values of an updated row by prefixing the column names with the :OLD and :NEW qualifiers.

Example

```
SQL>  DELETE FROM  employee WHERE
    emp_last_name = 'Joshi';
1 row deleted.
SQL> SELECT * FROM xemployee;
```

```
EMP_SSN    EMP_LAST_NAME
EMP_FIRST_NAME DELDATE
-------------  ------------------------    -----------
--------------- ------------------
999333333  Joshi                            Dinesh
02-MAY-03
```

ENABLING, DISABLING, DROPPING TRIGGERS

- SQL>ALTER TRIGGER trigger_name DISABLE;
- SQL>ALTER TABLE table_name DISABLE ALL TRIGGERS;
- **To enable a trigger, which is disabled, we can use the following syntax:**

  SQL>ALTER TABLE table_name ENABLE trigger_name;
- **All triggers can be enabled for a specific table by using the following command**
- SQL> ALTER TABLE table_name ENABLE ALL TRIGGERS;
- SQL> DROP TRIGGER trigger_name