PL/SQL: Database Triggers

INFS602 Physical Database Design

Learning Outcomes

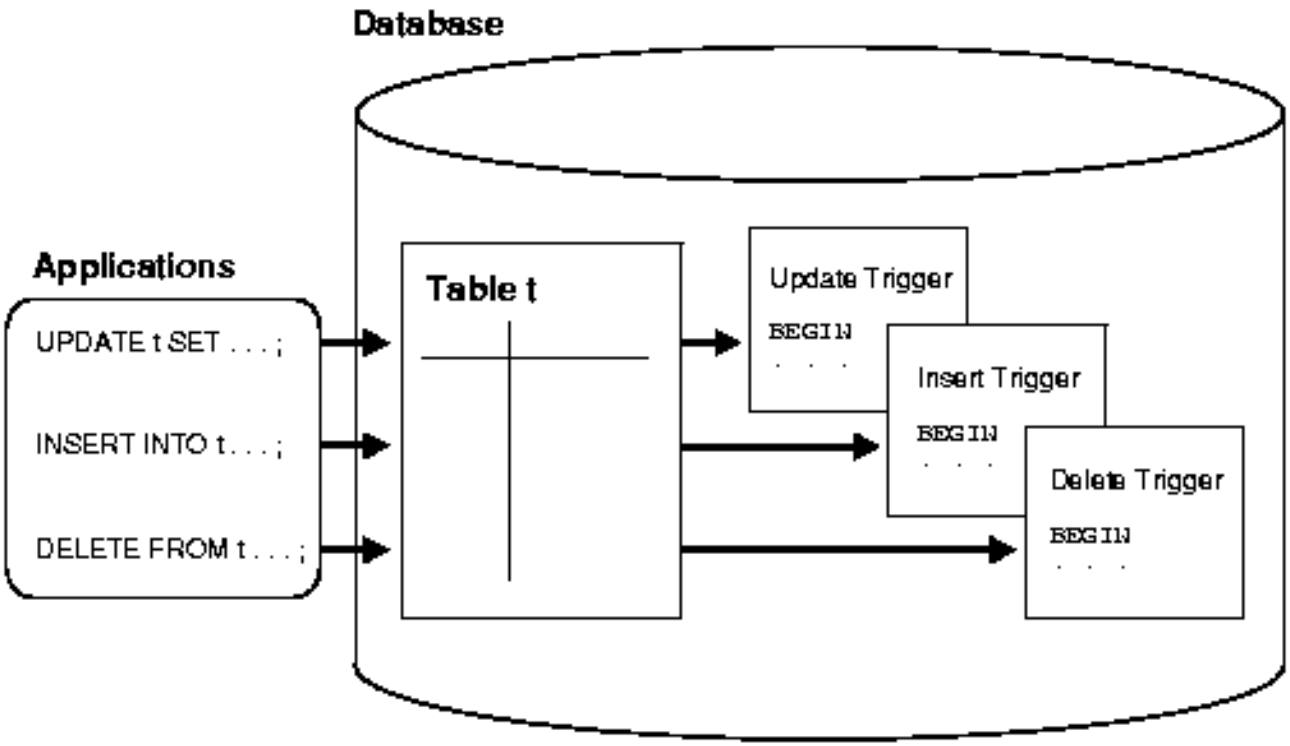
- Be able to write database triggers
- Understand the need for statement triggers and row triggers.
- Differentiate between statement and row level triggers

Database Triggers

- A trigger is a PL/SQL block that executes implicitly whenever a particular event takes place.

- A trigger can be either a database trigger or an application trigger.

Database Triggers



How Triggers Are Used

- Automatically generate derived column values
- Prevent invalid transactions
- Enforce complex security authorizations
- Enforce referential integrity across nodes in a distributed database
- Enforce complex business rules
- Provide transparent event logging
- Provide auditing
- Maintain synchronous table replicates
- Gather statistics on table access

Trigger Components

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Complete Syntax (for Table and Row)

- Trigger Timing
  - When should the trigger fire
  - BEFORE – The code in the trigger will execute before the triggering DML
  - AFTER – The code in the trigger body will execute after the triggering DML event
  - INSTEAD OF - The INSTEAD OF clause is used for creating trigger on a view

Trigger Components

- Trigger Event (DML operation)
  - Which data manipulation operation on the table causes the trigger to fire, INSERT, UPDATE or DELETE
- Trigger Type
  - How many times the trigger body executes
  - Statement: The trigger body executes once for the triggering event
  - ROW: The trigger body executes once for each row affected by the triggering event
- Trigger Body
  - What action the trigger performs
  - Defined with an anonymous PL/SQL block

Statement Triggers

- Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing event1[OR event2 OR event3]
ON table_name
PL/SQL block
```

- Example

```
CREATE or REPLACE TRIGGER log_emp_update
AFTER UPDATE ON Emp
BEGIN
    INSERT INTO emp_log (log_date, action)
        VALUES (SYSDATE, 'Emp table
          Changed');
END;
```

Row Triggers

- Syntax

```
CREATE [OR REPLACE] TRIGGER
  trigger_name
Timing event1 [OR event2 OR event3]
ON table_name
[REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW
[WHEN condition]
PL/SQL block;
```

- Example

  On next slide

Example

Main Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values

Trigger

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

1a

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

1b

```
Old salary:
New salary: 7500
Salary difference:
```

```
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

2a

2b

```
Old salary: 1500
New salary: 2000
Salary difference: 500
```

Row Trigger example

Audit Table

Username	TableName	Del	ColumnName
Jamie	EMP	3	
Jamie	EMP		SAL
Jamie	EMP		COMM
Jet	EMP		
Jet	EMP		SAL
Jet	EMP		COMM
Jamie	DEPT		

/*This trigger updates audit table to record each delete performed on Emp table */

```
CREATE OR REPLACE TRIGGER Emp_Delete
AFTER DELETE ON emp
FOR EACH ROW
BEGIN
        UPDATE audit_table SET del = del + 1
        WHERE username = user AND tablename = 'EMP'
        AND columnname IS NULL;
END;
/
```

Using Old and New Qualifiers

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN
   INSERT INTO audit_emp_values (user_name,
     timestamp, id, old_last_name, new_last_name,
     old_title, new_title, old_salary, new_salary)
   VALUES (USER, SYSDATE, :old.empno, :Old.ename,
     :new.ename, :old.job, :new.job, :old.sal,
     :new.sal);
END;
/
```

```
CREATE OR REPLACE TRIGGER audit_emp
AFTER DELETE OR INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN

IF DELETING THEN
        UPDATE audit_table SET del = del + 1
        WHERE user_name = user AND table_name = 'emp'
        AND column_name IS NULL;
  ELSIF INSERTING THEN
        UPDATE audit_table SET ins = ins + 1
        WHERE user_name = user AND table_name = 'emp'
        AND column_name IS NULL;
  ELSIF UPDATING ('SAL') THEN
        UPDATE audit_table SET upd = upd + 1
        WHERE user_name = user AND table_name = 'emp'
        AND column_name = 'SAL';
  ELSE /* general update*/
        UPDATE audit_table SET upd = upd + 1
        WHERE user_name = user AND table_name = 'emp'
        AND column_name IS NULL;
  END IF;
END;
/
```

Trigger Execution Model

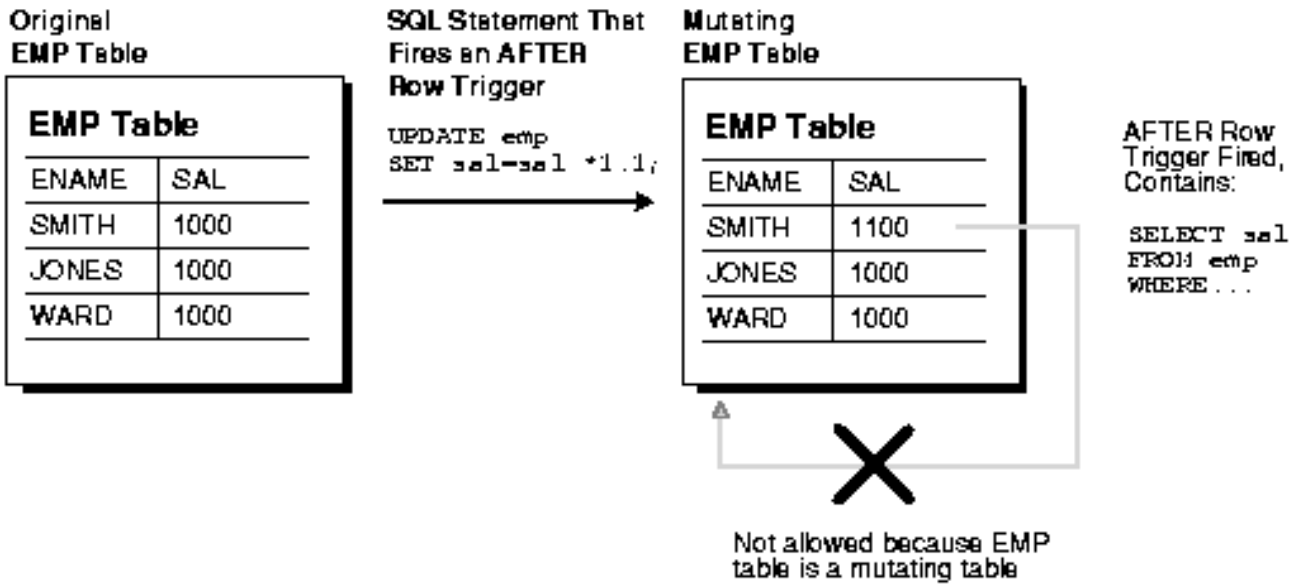
1. Execute all BEFORE STATEMENT triggers
2. Loop for each row affected
    1. Execute all BEFORE ROW triggers
    2. Execute the DML statement and perform integrity constraint checking
    3. Execute all AFTER ROW triggers
3. Complete deferred integrity constraint checking
4. Execute all AFTER STATEMENT triggers

Mutating Table

- A mutating table is a table that is currently being modified by an UPDATE, DELETE, or INSERT statement, or a table that might need to be updated by the effects of a declarative DELETE CASCADE referential integrity action
- The triggered table itself is a mutating table

Mutating Table

- The SQL statements of a trigger cannot read from (query) or modify a mutating table of the triggering statement



```
CREATE OR REPLACE TRIGGER check_salary
BEFORE INSERT OR UPDATE of sal, job ON emp
FOR EACH ROW
WHEN (new.job <> 'PRESIDENT)

DECLARE
    v_minsal emp.sal%TYPE;
    v_maxsal emp.sal%TYPE;
BEGIN
    SELECT MIN(sal), MAX(sal)
    INTO v_minsal, v_maxsal
    FROM emp
    WHERE job = :new.job;
    IF :new.sal < v_minsal OR
        :new.sal > v_maxsal THEN
        RAISE_APPLICATION_ERROR (-20505, 'Out of
          range');
    END IF;
END;
/
```

- The EMP table is mutating, or in a state of change, therefore the trigger cannot read from it

References

- *Oracle 11g* PL/SQL User's Guide and Reference
- *Oracle 11g* Application Developer's Guide—Fundamentals
  - See Chapter 7 Coding PL/SQL Procedures and Packages
  - See Chapter 9 Coding Triggers
- Database Systems Using ORACLE, A Simplified Guide to SQL and PL/SQL, 2nd edition, Nilesh Shah

Important points are needed to be considered

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.