GALGOTIAS UNIVERSITY

School of Computing Science and Engineering

Program: BCA - IOP

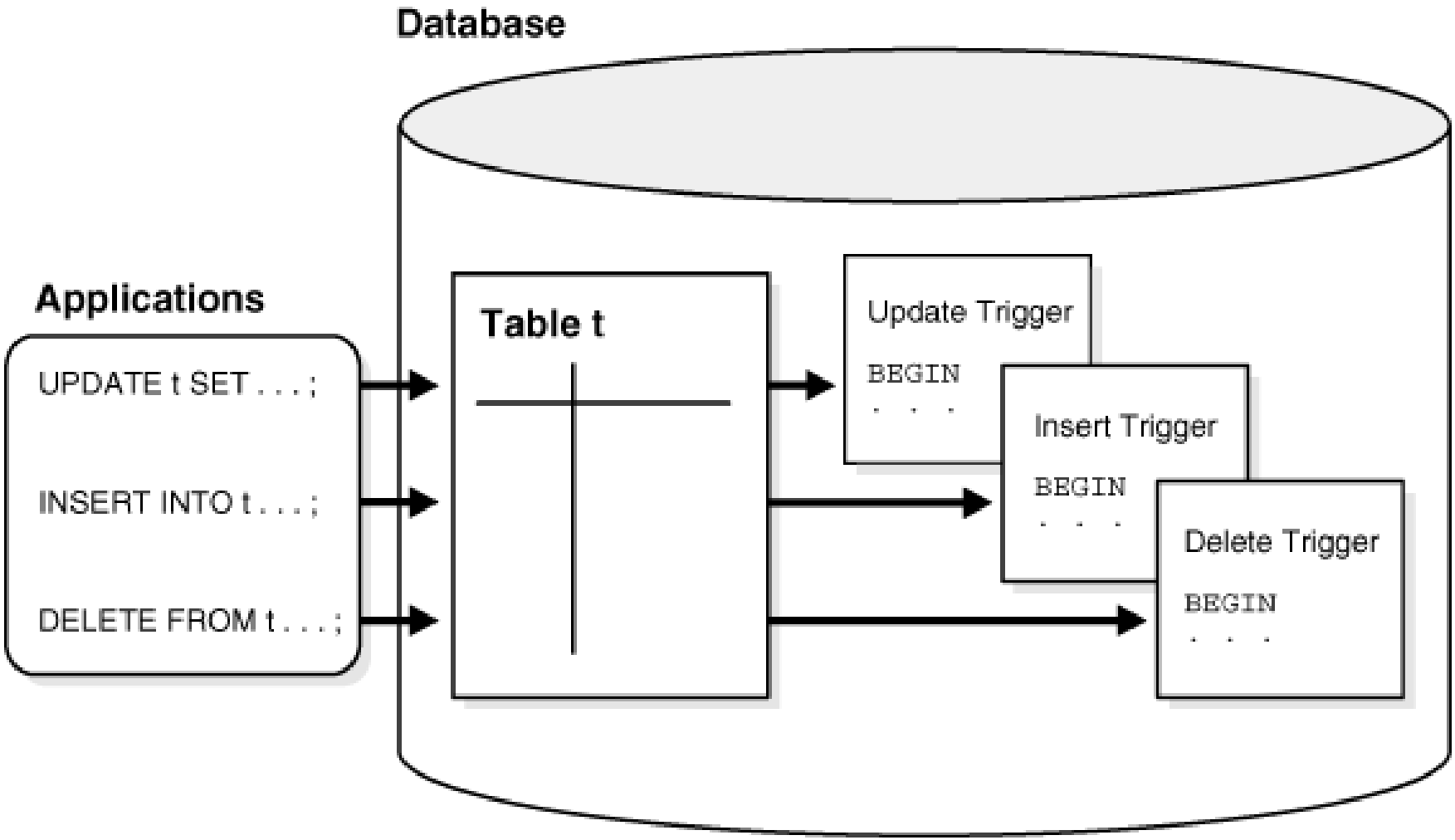
Course Code: BCAS3031

Course Name: PL/SQL & Cursors and Triggers

Dr. T. Poongodi

Associate Professor

- TRIGGERS are stored programs that are fired by Oracle engine automatically when DML Statements like insert, update, delete are executed on the table or some events occur.
- The code to be executed in case of a trigger can be defined as per the requirement.
- The purpose of trigger is to maintain the integrity of information on the database.


Database
Applications
UPDATE t SET . . . ;
INSERT INTO t . . . ;
DELETE FROM t . . . ;
Table t
Update Trigger
BEGIN
. . .
Insert Trigger
BEGIN
. . .
Delete Trigger
BEGIN
. . .

Triggers are stored programs, which are automatically executed or fired when some events occur.

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Types of Triggers

Classification based on the **timing**

- BEFORE Trigger: It fires before the specified event has occurred.
- AFTER Trigger: It fires after the specified event has occurred.
- INSTEAD OF Trigger: A special type. (only for DML)

Classification based on the **level**

- STATEMENT level Trigger: It fires one time for the specified event statement.
- ROW level Trigger: It fires for each record that got affected in the specified event. (only for DML)

Classification based on the **Event**

- DML Trigger: It fires when the DML event is specified (INSERT/UPDATE/DELETE)
- DDL Trigger: It fires when the DDL event is specified (CREATE/ALTER)
- DATABASE Trigger: It fires when the database event is specified (LOGON/LOGOFF/STARTUP/SHUTDOWN)

Type of Triggers

1. BEFORE Trigger: BEFORE trigger execute before the triggering DML statement (INSERT, UPDATE, DELETE) execute. Triggering SQL statement is may or may not execute, depending on the BEFORE trigger conditions block.
2. AFTER Trigger: AFTER trigger execute after the triggering DML statement (INSERT, UPDATE, DELETE) executed. Triggering SQL statement is execute as soon as followed by the code of trigger before performing Database operation.
3. ROW Trigger: ROW trigger fire for each and every record which are performing INSERT, UPDATE, DELETE from the database table. If row deleting is define as trigger event, when trigger file, deletes the five rows each times from the table.
4. Statement Trigger: Statement trigger fire only once for each statement. If row deleting is define as trigger event, when trigger file, deletes the five rows at once from the table.

Type of Triggers

1. Combination Trigger: Combination trigger are combination of two trigger type,
    1. Before Statement Trigger: Trigger fire only once for each statement before the triggering DML statement.
    2. Before Row Trigger : Trigger fire for each and every record before the triggering DML statement.
    3. After Statement Trigger: Trigger fire only once for each statement after the triggering DML statement executing.
    4. After Row Trigger: Trigger fire for each and every record after the triggering DML statement executing.

```
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;
```

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.

- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Select * from customers;

```
+----+---------+-----+----------+---------+
| 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 |
+----+---------+-----+----------+---------+
```

- 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 −

```
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;
/
```

Trigger created.

- OLD and NEW references are not available for table-level triggers, rather use for record-level triggers.
- To query the table in the same trigger, 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, write 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.

Triggering a Trigger

Perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

```
INSERT INTO CUSTOMERS
(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, ‘Kavin', 22, ‘TN’, 42000.00 );
```

When a record is created in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

Old salary:

New salary: 7500

Salary difference:

Because this is a new record, old salary is not available and the above result comes as null.

Another DML operation on the CUSTOMERS table.

The UPDATE statement will update an existing record in the table −

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

When a record is updated in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

Old salary: 1500

New salary: 2000

Salary difference: 500

```
SQL> create table customers(ID number,NAME
varchar2(20),AGE number,ADDRESS
varchar2(20),SALARY number(8,2));

Table created.
```

```
SQL> insert into customers values(1,'Anu',22,'Haryana',45000);

1 row created.

SQL> insert into customers values(1,'Akil',21,'Delhi',35000);

1 row created.

SQL> insert into customers values(1,'Nikil',21,'UP',25000);

1 row created.
```

SQL> select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Anu	22	Haryana	45000
1	Akil	21	Delhi	35000
1	Nikil	21	UP	25000

```
SQL> CREATE OR REPLACE TRIGGER display_salary_changes
  2  BEFORE DELETE OR INSERT OR UPDATE ON customers
  3  FOR EACH ROW
  4  WHEN (NEW.ID > 0)
  5  DECLARE
  6     sal_diff number;
  7  BEGIN
  8     sal_diff := :NEW.salary  - :OLD.salary;
  9     dbms_output.put_line('Old salary: ' || :OLD.salary);
 10     dbms_output.put_line('New salary: ' || :NEW.salary);
 11     dbms_output.put_line('Salary difference: ' || sal_diff);
 12  END;
 13  /

Trigger created.
```

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

1 row created.

SQL> select * from customers;

        ID NAME                 AGE ADDRESS                  SALARY
---------- ------------------ ---------- -------------------- ----------
         1 Anu                        22 Haryana                   45000
         1 Akil                       21 Delhi                     35000
         1 Nikil                      21 UP                        25000
         7 Kriti                      22 HP                         7500
```

```
SQL> UPDATE customers SET salary = salary + 500 WHERE id = 7;

1 row updated.

SQL> select * from customers;

        ID NAME                 AGE ADDRESS              SALARY
---------- -------------------- --------- -------------------- ----------
         1 Anu                   22 Haryana                45000
         1 Akil                  21 Delhi                  35000
         1 Nikil                 21 UP                     25000
         7 Kriti                 22 HP                     8000

SQL>
```

Run SQL Command Line

```
1 row created.

SQL> insert into customers values(1,'Nikil',21,'UP',25000);

1 row created.

SQL> select * from customers;

        ID NAME                      AGE ADDRESS                SALARY
---------- -------------------- ---------- -------------------- ----------
         1 Anu                          22 Haryana                 45000
         1 Akil                         21 Delhi                   35000
         1 Nikil                        21 UP                      25000

SQL> CREATE OR REPLACE TRIGGER display_salary_changes
  2  BEFORE DELETE OR INSERT OR UPDATE ON customers
  3  FOR EACH ROW
  4  WHEN (NEW.ID > 0)
  5  DECLARE
  6     sal_diff number;
  7  BEGIN
  8     sal_diff := :NEW.salary  - :OLD.salary;
  9     dbms_output.put_line('Old salary: ' || :OLD.salary);
 10     dbms_output.put_line('New salary: ' || :NEW.salary);
 11     dbms_output.put_line('Salary difference: ' || sal_diff);
 12  END;
 13  /

Trigger created.

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

1 row created.

SQL> select * from customers;

        ID NAME                      AGE ADDRESS                SALARY
---------- -------------------- ---------- -------------------- ----------
         1 Anu                          22 Haryana                 45000
         1 Akil                         21 Delhi                   35000
         1 Nikil                        21 UP                      25000
         7 Kriti                        22 HP                       7500
```

1) BEFORE UPDATE, Statement Level: This trigger will insert a record into the table 'product_check' before a sql update statement is executed, at the statement level.

```
CREATE or REPLACE TRIGGER Before_Update_Stat_product
BEFORE
UPDATE ON product
Begin
INSERT INTO product_check
Values('Before update, statement level',sysdate);
END;
/
```

2) BEFORE UPDATE, Row Level: This trigger will insert a record into the table 'product_check' before each row is updated.

```
CREATE or REPLACE TRIGGER Before_Update_Row_product
BEFORE
UPDATE ON product
FOR EACH ROW
BEGIN
INSERT INTO product_check
Values('Before update row level',sysdate);
END;
/
```

3) AFTER UPDATE, Statement Level: This trigger will insert a record into the table 'product_check' after a sql update statement is executed, at the statement level.

```
CREATE or REPLACE TRIGGER After_Update_Stat_product
AFTER
UPDATE ON product
BEGIN
INSERT INTO product_check
Values('After update, statement level', sysdate);
End;
/
```

4) AFTER UPDATE, Row Level: This trigger will insert a record into the table 'product_check' after each row is updated.

```
CREATE or REPLACE TRIGGER After_Update_Row_product
AFTER
insert On product
FOR EACH ROW
BEGIN
INSERT INTO product_check
Values('After update, Row level',sysdate);
END;
/
```

Execute a update statement on table product.

```
UPDATE PRODUCT SET unit_price = 800
WHERE product_id in (100,101);
```

```
DROP TRIGGER trigger_name;
```


Thank You