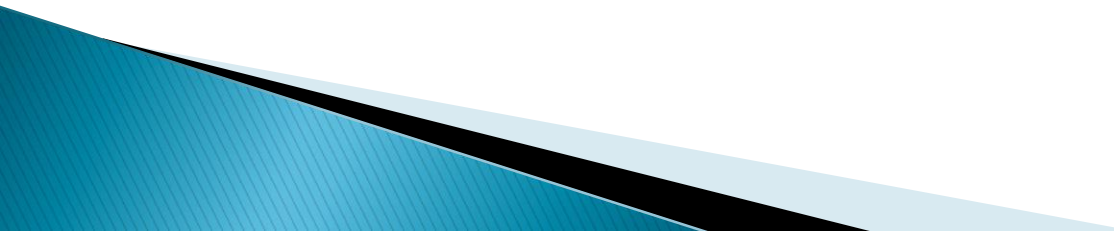



Database Triggers

Md. Tohidul Islam
Associate Professor
Dept. of Computer Science & Engineering
University of Rajshahi

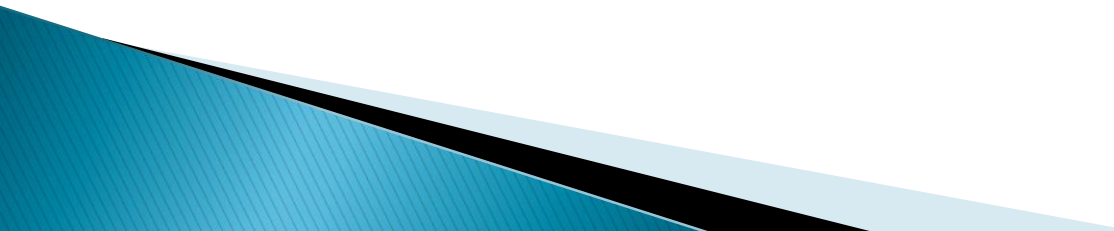
Database Triggers

- ▶ Database Triggers are procedures that are stored in the database and are implicitly executed(fired) when the contents of a table are changed (due to insert, update, delete).
 - ▶ These procedures are fired automatically by Oracle itself.
- 

Use of Triggers

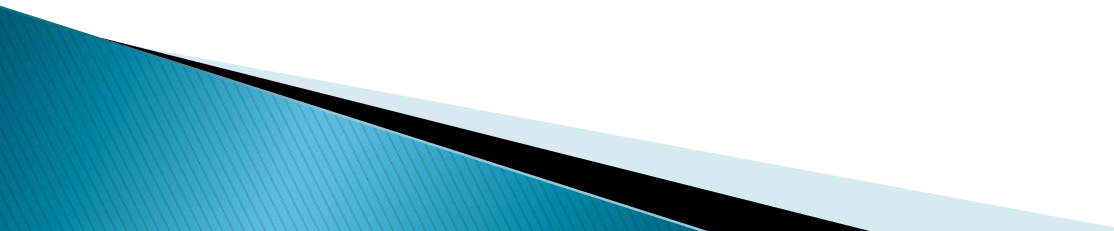
- ▶ A trigger can permit DML statements against a table only if they are issued, during regular business hours or on predetermined weekdays.
 - ▶ Can be used to keep an audit trail of a table (to store the modified and deleted records of the table) along with the operation performed and the time on which the operation was performed.
 - ▶ Prevent invalid transactions.
 - ▶ Enforces complex security authorizations.
- 

Note about Trigger

- ▶ When a triggered is fired, a SQL statement inside the trigger can also fire the same or some other trigger(cascading).
 - ▶ Excessive use of triggers for customizing the database can result in complex interdependence between the triggers, which may be difficult to maintain in a large application.
- 

How to Apply Database Triggers

- ▶ A trigger has three basic parts:
 1. Triggering event or statement.
 2. Trigger restriction.
 3. Trigger action.

 - ▶ Triggering event or statement:
 - It is a SQL statement that causes a trigger to be fired.
 - It can be insert, update or delete statement for a specific table.
- 

Parts of Triggers

▶ Trigger restriction:

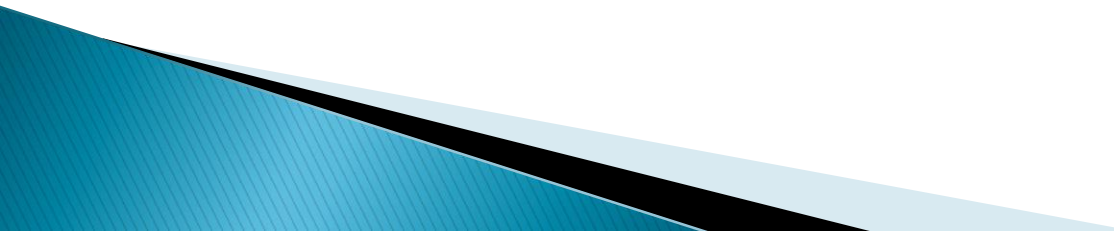
- A trigger restriction specifies a Boolean (logical) expression that must be true for the trigger to fire.
- Trigger restriction is specified using a when clause.

▶ Trigger action.

- Trigger action is the procedure(PL/SQL block) that contains the SQL statement to be executed when a triggering statement is issued and the trigger restriction evaluates to true.
- For row triggers, the statements in a trigger action have access to column values (new and old) of the current row being processed.

Types of Triggers

▶ Row Triggers:

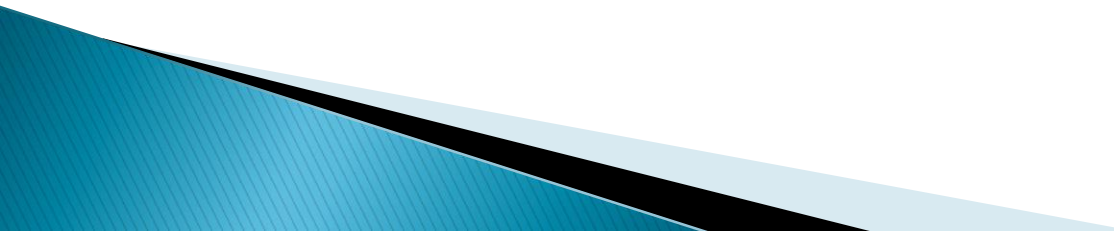
- A row trigger is fired each time the table is affected by the triggering statement.
 - Example: if an update statement updates multiple row of a table, a row trigger is fired once for each row affected by the update statement.
 - if it affects no records, the trigger is not executed at all.
 - i.e. row trigger keeps track of all affected records.
- 

Types of Triggers

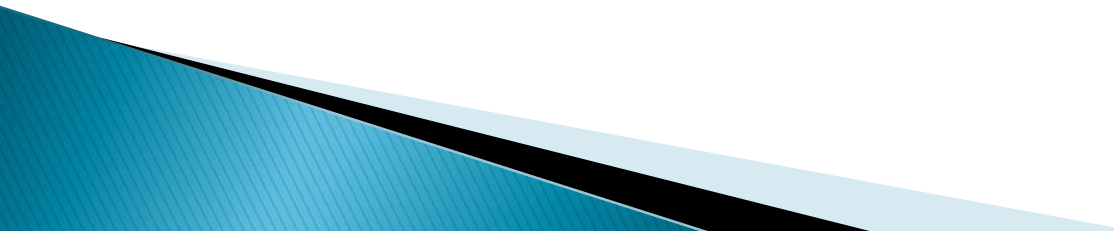
▶ Statement Triggers:

- A statement trigger is fired once on behalf of the triggering statement, independent of the number of rows the triggering statement affects (even if no record are affected).
- i.e. statement trigger makes the security check on the time or the user.

Types of Triggers

- ▶ Before VS. After Triggers:
 - You can specify the trigger timing.
 - i.e. you can specify when triggering action is to be executed in relation to the triggering statement.
 - Before and After apply to both row and the statement triggers.
 - Before triggers are used when the trigger action should determine whether or not the triggering statement should be allowed to complete.
 - After triggers are used when you want the triggering statement to complete before executing the trigger action.
- 

Types of Triggers

- ▶ Using the options explained above , four types of triggers can be created.
 1. Before statement trigger:
 2. Before row trigger:
 3. After statement trigger:
 4. After row trigger:
- 

Syntax for Creating Trigger

```
CREATE OR REPLACE TRIGGER [schema] triggerName  
    {BEFORE, AFTER}  
    {DELETE, INSERT, UPDATE [OF column,...]}  
    ON [schema] tableName  
    [REFERENCING {OLD AS old, NEW AS new}]  
    [FOR EACH ROW [WHEN condion]]
```

DECLARE

Variable declaration;
Constant declaration;

BEGIN

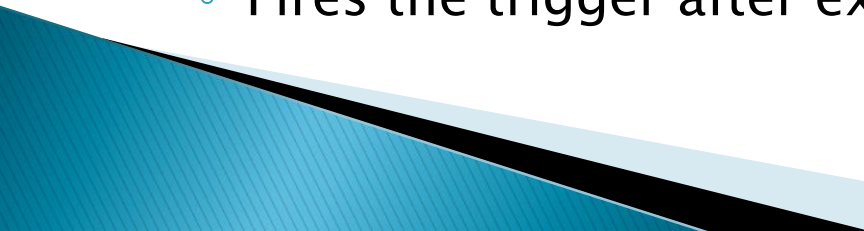
PL/SQL subprogram body;

EXCEPTION

Exception PL/SQL block;

END

Keywords and Parameters

- ▶ **OR REPLACE:**
 - Recreates the trigger if it already exists.
 - ▶ **Schema:**
 - Is the schema to contain the trigger. Oracle creates the trigger in your own schema(If schema omitted).
 - ▶ **triggerName:**
 - Name of the trigger to be created.
 - ▶ **BEFORE:**
 - Fires the trigger before executing the triggering statement.
 - ▶ **AFTER:**
 - Fires the trigger after executing the triggering statement.
- 

Keywords and Parameters

- ▶ **DELETE:**
 - Fires the trigger whenever a DELETE statement removes a row from the table.
- ▶ **INSERT:**
 - Fires the trigger whenever an INSERT statement adds a row to the table.
- ▶ **UPDATE:**
 - Fires the trigger whenever an UPDATE statement changes a value in one of the columns specified in the OF clause.
 - If you omit the OF clause Oracle fires if any change in any column.
- ▶ **ON:**
 - Specifies the schema and name of the table on which the trigger is to be created.
 - If you omit schema, Oracle assumes the table is in your own schema.
 - You can not create a trigger on a table in the schema SYS.

Keywords and Parameters

▶ REFERENCING:

- Specifies correlation name.
- You can use these name in the PL/SQL block and when clause of a row trigger to refer specifically to old and new values of the current row.

▶ FOR EACH ROW:

- Oracle fires a row trigger once for each row.
- If you omit this clause, the trigger is a statement trigger.

▶ WHEN:

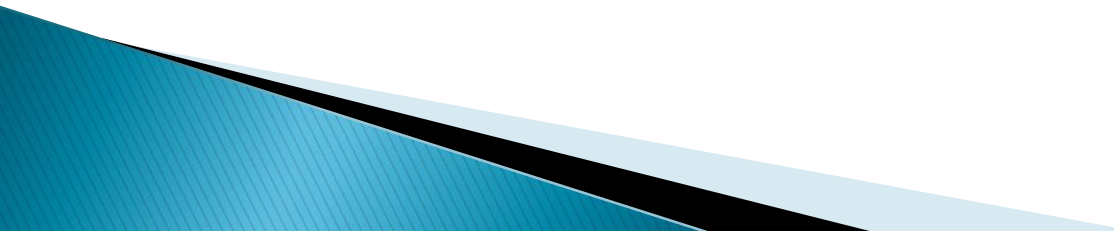
- Specifies the trigger restriction.
- Trigger restriction is a SQL condition that must be satisfied for Oracle to fire.

Trigger Example 01

Ex: Write a trigger that checks that balance of table account(aname, anumber, balance) does not become negative.

```
CREATE TRIGGER check_balance  
  BEFORE UPDATE OF balance  
  ON account  
  FOR EACH ROW
```

```
DECLARE  
  New_bal number(10,2);          /*Variable that hold new balance*/  
  
BEGIN  
  New_bal:= :new.balance;        /*Assigning the new balance*/  
  IF New_bal<0 THEN  
    raise_application_error(-20001,'Balance can not be negative');  
  End iF;  
END;
```



Trigger Example 02

Ex: Write a trigger that checks any update or delete of table data account(aname, anumber, balance) and stores old data, the date and operation into an audit table auditaccount.

```
CREATE TRIGGER audit_trail
  AFTER UPDATE OR DELETE ON account
  FOR EACH ROW
```

```
DECLARE
  balance number(10,2);           /*Variable that hold new balance*/
  oper varchar2(10);
  aname varchar2(20);
  anumber number(10,2);
```

```
BEGIN
  if updating then      oper:='Update'; end if;
  if deleting then      oper:='Delete'; end if;
  balance:= :old.balance; /*Assigning the old balance*/
  aname:= :old.aname;
  anumber:= :old.anumber;

  Insert into auditaccount
  Values(aname, anumber, balance, oper, sysdate);
END;
```