Course Code: CL-2005	Course : Database Systems Lab
Instructor(s) :	Sohail Ahmed Malik

Contents: In this lab, we will discuss Triggers in PL/SQL.
Triggers are named PL/SQL blocks which are stored in the database. We can also say that they are specialized stored programs which execute implicitly when a triggering event occurs. This means we cannot call and execute them directly instead they only get triggered by events in the database.

Events Which Fires the Database Triggers
These events can be anything such as

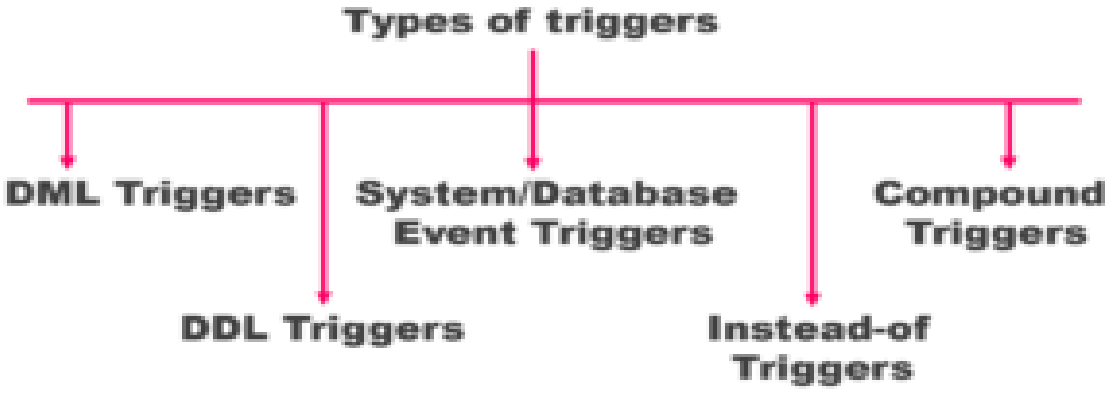
A DML Statement – An Update, Insert or Delete statement executing on any table of your database. You can program your trigger to execute either BEFORE or AFTER executing your DML statement. For example, you can create a trigger which will get fired *Before* the Update. Similarly, you can create a trigger which will get triggered after the execution of your INSERT DML statement.

A DDL Statement – Next type of triggering statement can be a DDL Statement such as CREATE or ALTER. These triggers can also be executed either BEFORE or AFTER the execution of your DDL statement. These triggers are generally used by DBAs for auditing purposes. And they really come in handy when you want to keep an eye on the various changes on your schema. For instance, who created the object or which user. Just like some cool spy tricks.

A system event. – Yes, you can create a trigger on a system event. And by a system event, I mean shut down or startup of your database.

A User Events – Another type of triggering event can be User Events such as log off or log on onto your database. You can create a trigger which will either execute before or after the event. Furthermore, it will record the information such as time of event occur, the username who created it.

Types of Database Triggers
There are 5 types of triggers in the Oracle database. 3 of them are based on the triggering event which are discussed in the previous section.



Data Manipulation Language Triggers or DML triggers

As the name suggests these are the triggers which depend on DML statements such as Update, Insert or Delete. They get fired either before or after them. Using DML trigger you can control the behavior of your DML statements. You can audit, check, replace or save values before they are changed. Automatic Increment of your Numeric primary key is one of the most frequent tasks of these types of triggers.

Data Definition Language Triggers or DDL triggers.

Again as the name suggests these are the type of triggers which are created over DDL statements such as CREATE or ALTER. They get fired either before or after the execution of your DDL statements. Using this type of trigger you can monitor the behavior and force rules on your DDL statements.

System or Database Event triggers.

Third type of triggers is system or database triggers. These are the type of triggers which come into action when some system event occurs such as database log on or log off. You can use these triggers for auditing purposes. For example, keeping an eye on information of system access like say who connects with your database and when. Most of the time System or Database Event triggers work as Swiss Knife for DBAs and help them in increasing the security of the data.

Instead-of Trigger

This is a type of trigger which enables you to stop and redirect the performance of a DML statement. Often this type of trigger helps you in managing the way you write to non-updatable views. You can also see the application of business rules by INSTEAD OF triggers where they insert, update or delete rows directly in tables that are defining updatable views. Alternatively, sometimes the INSTEAD OF triggers are also seen inserting, updating or deleting rows in designated tables that are otherwise unrelated to the view.

Compound triggers

These are multi-tasking triggers that act as both statement as well as row-level triggers when the data is inserted, updated or deleted from a table. You can capture information at four timing points using this trigger:

- before the firing statement;
- prior to the change of each row from the firing statement; □ post each row changes from the firing statement;
- After the firing statement.

All these types of triggers can be used to audit, check, save and replace the values. Even before they are changed right when there is a need to take action at the statement as well as at row event levels.

The Syntax of Database Trigger

```
CREATE [OR REPLACE] TRIGGER Ttrigger_name
{BEFORE|AFTER} Triggering_event ON table_name
[FOR EACH ROW]
[FOLLOWS another_trigger_name]
[ENABLE/DISABLE]
[WHEN condition] DECLARE
  declaration statements
BEGIN
  executable statements
EXCEPTION
  exception-handling statements
END;
```

Uses of Database triggers.

Using database triggers we can enforce business rules that can't be defined by using integrity constants.
Using triggers we can gain strong control over the security.
We can also collect statistical information on the table access.
We can automatically generate values for derived columns such as auto increment numeric primary key.
Using database triggers we can prevent the invalid transactions.

Data Manipulation Language (DML) Triggers.

As the name suggests these are the triggers which execute on DML events or say depend on DML statements such as Update, Insert or Delete. Using DML trigger you can control the behavior of your DML statements.

Examples
In order to demonstrate the creation process of DML trigger we need to first create a table.

CREATE TABLE superheroes (sh_name VARCHAR2 (15));

I have created this table with the name SUPERHEROES which has only one column sh_name with data type varchar2 and data width 15. Now I will write a DML trigger which will work on this table.

Example 1. Before Insert Trigger

In the first example we will see how to create a trigger over Insert DML. This trigger will print a user defined message every time a user inserts a new row in the superheroes table.

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER bi_Superheroes
BEFORE INSERT ON superheroes
FOR EACH ROW
ENABLE
DECLARE
  v_user  VARCHAR2 (15);
BEGIN
 SELECT user INTO v_user FROM dual;
 DBMS_OUTPUT.PUT_LINE('You Just Inserted a Row Mr.'|| v_user);  END;/
```

Example 2: Before Update Trigger.

Update Trigger is the one which will execute either before or after Update DML. The creation process of an Update trigger is the same as that of Insert Trigger. You just have to replace Keyword INSERT with UPDATE in the 2nd Line of the above example.

```
CREATE OR REPLACE TRIGGER bu_Superheroes
BEFORE UPDATE ON superheroes
FOR EACH ROW
ENABLE
DECLARE
  v_user  VARCHAR2 (15);
BEGIN
 SELECT user INTO v_user FROM dual;
 DBMS_OUTPUT.PUT_LINE('You Just Updated a Row Mr.'|| v_user);
END;
/
```

Example 3: Before Delete Trigger

Similar to Insert and Update DML you can write a trigger over Delete DML. This trigger will execute either before or after a user deletes a row from the underlying table.

```
CREATE OR REPLACE TRIGGER bu_Superheroes
BEFORE DELETE ON superheroes
FOR EACH ROW
ENABLE
DECLARE
  v_user  VARCHAR2 (15);
BEGIN
 SELECT user INTO v_user FROM dual;
 DBMS_OUTPUT.PUT_LINE('You Just Deleted a Row Mr.'|| v_user);
END;
/
```

INSERT, UPDATE, DELETE All in One DML Trigger Using IF-THEN-ELSIF

```
CREATE OR REPLACE TRIGGER tr_superheroes
BEFORE INSERT OR DELETE OR UPDATE ON superheroes
FOR EACH ROW
ENABLE
DECLARE
  v_user  VARCHAR2(15);
BEGIN
  SELECT
    user INTO v_user FROM dual;
```

```
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('one line inserted by '||v_user);
  ELSIF DELETING THEN
    DBMS_OUTPUT.PUT_LINE('one line Deleted by '||v_user);
  ELSIF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('one line Updated by '||v_user);
  END IF;
END;
/
```

Table Auditing

Table auditing means keeping a track of all the dml activities performed on a specific table of the database for example which user Inserted, updated or deleted a row from the table and when. It is like spying on the users who are messing your table's data.

Example

For the demonstration we will use the table 'Superheroes' which we created in the previous tutorial.

Suppose you want to keep an eye on the users who are inserting, updating or deleting data from the 'Superheroes' table. Let's see how we can achieve this. To do so we will need another table in which we can journal the auditing data entries.

```
CREATE    TABLE    sh_audit(
new_name          varchar2(30),
old_name          varchar2(30),
user_name         varchar2(30),
entry_date        varchar2(30),
operation  varchar2(30)
);
```

This table sh_audit has 5 columns which will store the auditing information such as the new data inserted or updated, old data which is updated or deleted from the table, name of the user who did it along with the date and time, also the type of DML operation performed.

Write a trigger on the source table superheroes and will store the data into the auditing table sh_audit.

```
CREATE OR REPLACE trigger superheroes_audit
BEFORE INSERT OR DELETE OR UPDATE ON superheroes
FOR EACH ROW
ENABLE
DECLARE
  v_user  varchar2  (30);       v_date
varchar2(30);
BEGIN
  SELECT user, TO_CHAR(sysdate, 'DD/MON/YYYY HH24:MI:SS') INTO v_user, v_date FROM dual;
  IF INSERTING THEN
    INSERT INTO sh_audit (new_name,old_name, user_name, entry_date, operation)
VALUES(:NEW.SH_NAME, Null , v_user, v_date, 'Insert');
  ELSIF DELETING THEN
    INSERT INTO sh_audit (new_name,old_name, user_name, entry_date, operation)
    VALUES(NULL,:OLD.SH_NAME, v_user, v_date, 'Delete');
  ELSIF UPDATING THEN
    INSERT INTO sh_audit (new_name,old_name, user_name, entry_date, operation)
VALUES(:NEW.SH_NAME, :OLD.SH_NAME, v_user, v_date,'Update');   END IF;
END;
/
```

Pseudo Records (New/Old)

These Psuedo Records helps us in fetching data from the sh_name column of the underlying source table 'Superheroes' and storing it into the audit table sh_audit.

Pseudo Record ': NEW', allows you to access a row currently being processed. In other words, when a row is being inserted or updated into the superheroes table. Whereas Pseudo Record ': OLD' allows you to access a row which is already being either Updated or Deleted from the superheroes table.

In order to fetch the data from the source table, you have to first write the proper Pseudo Record (New/Old) followed by dot (.) and the name of the column of the source table whose value you want to fetch. For example in our case we want to fetch the data from sh_name column which belongs to our source table superheroes. Thus we will write ": New. sh_name" for fetching the current value and to fetch the previously stored value we will write ": OLD. sh_name". Once the values are fetched the INSERT dml will store these values into the respective columns of the audit table.

Restriction on Pseudo Record

- For an INSERT trigger, OLD contain no values, and NEW contain the new values.
- For an UPDATE trigger, OLD contain the old values, and NEW contain the new values.
- For a DELETE trigger, OLD contain the old values, and NEW contain no values.

DML Pseudo Records	Insert	Update	Delete
New	✓	✓	✗
Old	✗	✓	✓

Once you execute and compile this trigger then you can take it on a test run by writing DML statements on the underlying source table 'Superheroes'. For example you can try Inserting a row in superheroes table and then check the audit table whether there is some data or not.

Synchronized backup copy of a table.
The backup table gets automatically populated or updated with the main table simultaneously.

For the demonstration we will require two identical tables; one which will serve as your main table that will accept the data from your database user and the second which will be your backup table. I will use the Superheroes table which we have been using since the beginning of this DML trigger series as our main table.

CREATE TABLE superheroes (Sh_name VARCHAR2(30));

Next we will have to create an identical table to this one which will work as our backup table.

Let's create this backup table.
CREATE TABLE superheroes_backup AS SELECT * FROM superheroes WHERE 1=2;

The above command will create the identical table just like the main table superheroes only without data.
Next we have to write the trigger which will insert, update or delete the rows from the backup table when someone does the same with our main table.

```
CREATE or REPLACE trigger Sh_Backup
BEFORE INSERT OR DELETE OR UPDATE ON superheroes
FOR EACH ROW
ENABLE
BEGIN
  IF INSERTING THEN
    INSERT INTO superheroes_backup (SH_NAME) VALUES (:NEW.SH_NAME);
  ELSIF DELETING THEN
    DELETE FROM superheroes_backup WHERE SH_NAME =:old.sh_name;
  ELSIF UPDATING THEN
    UPDATE superheroes_backup
    SET SH_NAME =:new.sh_name WHERE SH_NAME =:old.sh_name;   END IF;
END;
/
```

Schema & Database Auditing Using DDL Trigger In PL/SQL

DDL triggers are the triggers which are created over DDL statements such as CREATE, DROP or ALTER. Using this type of trigger you can monitor the behavior and force rules on your DDL statements.

In order to proceed ahead and start writing the trigger first we need a table in which we can journal the auditing information created by the trigger.

```
CREATE TABLE schema_audit
 ( ddl_date      DATE, ddl_user       VARCHAR2(15), object_created VARCHAR2(15),     object_name
VARCHAR2(15), ddl_operation  VARCHAR2(15)
 );
```

In case of schema/user auditing using DDL trigger creates this table in the same schema which you are auditing and in case of Database auditing using DDL trigger create this table in sys or system schema (sys or system both schemas can be used to perform database auditing).

DDL Trigger for Schema Auditing

First you need to log on to the database using the schema which you want to audit. For example suppose you want to create the DDL trigger to audit the HR schema then log on to your database using the HR schema.

Then Write, Execute and Compile the below trigger.

```
CREATE OR REPLACE TRIGGER hr_audit_tr
AFTER DDL ON SCHEMA
BEGIN
   INSERT INTO schema_audit VALUES ( sysdate,
sys_context('USERENV','CURRENT_USER'),  ora_dict_obj_type,  ora_dict_obj_name,
ora_sysevent);
END;
/
```

If you will notice carefully the second line of the code (“AFTER DDL ON SCHEMA”) indicates that this trigger will work on the schema in which it is created. On successful compilation this trigger will insert the respective information such as the date when the DDL is executed, username who executed the DDL, type of database object created, name of the object given by the user at the time of its creation and the type of DDL into the table which we created earlier.

DDL Trigger for Database Auditing.

Similar to the schema auditing with some minor changes in the above trigger you can audit your database too. But for that first you need to logon to the database using either SYS user or SYSTEM user.

After doing that you have to create the above shown table under the same user so that your trigger can dump the auditing data without any read and write errors.

```
CREATE OR REPLACE TRIGGER db_audit_tr
AFTER DDL ON DATABASE
BEGIN
  INSERT INTO schema_audit VALUES (sysdate, sys_context('USERENV','CURRENT_USER'), ora_dict_obj_type, ora_dict_obj_name, ora_sysevent); END;/
```

If you notice the second line of this code carefully then you will find that we have replaced the keyword Schema with the keyword Database which indicates that this trigger will work for the whole database and will perform the underlying work.

To create a trigger on database we require ADMINISTER DATABASE TRIGGER system privilege. All the administrative users such as sys or system already has these privileges by default that is the reason we created this database auditing DDL trigger using these users. Though you can create the same trigger with any user by granting the same privileges to them but that is not advisable because of your database security reasons.

Schema Level Database LOGON Trigger In PL/SQL

Database event triggers also known as system event triggers come into action when some system event occurs such as database log on, log off, start up or shut down. These types of triggers are majorly used for monitoring activity of the system events and have been proved quite a powerful tool for a DBA.

Types of Database Event Triggers.

1. Schema Level Event Triggers
2. Database Level Event Triggers

Schema level event triggers can work on some specific schemas while the database event triggers have database wide scope. In other words database event triggers can be created to monitor the system event activities of either a specific user/schema or the whole database.

Object/System Privileges

Schema level event triggers can be created by any user of your database who has CREATE TRIGGER system privilege while the database event trigger can only be created by privileged user such as SYS or SYSTEM who has ‘Administrative Database Trigger’ System Privileges.

Syntax

```
CREATE OR REPLACE TRIGGER trigger_name
BEFORE | AFTER database_event ON database/schema
BEGIN
        PL/SQL Code
END;
/
```

Example. Schema Level Event Trigger.

Suppose user HR is a control freak and wants to monitor its every log on and log off activity. In this case what HR can do is, create event triggers on Log on and log off database event in its own schema.

Step 1: Connect to the database
Step 2: Create a Table

```
CREATE TABLE hr_evnt_audit
  ( event_type VARCHAR2(30), logon_date DATE, logon_time VARCHAR2(15),     logof_date DATE,
logof_time VARCHAR2(15)  );
```

Step3: Write the trigger Logon Schema Event Trigger.
This trigger will fire every time HR user logs on to the database and respective values will be stored into the table which we just created in the step 2.

```
CREATE OR REPLACE TRIGGER hr_lgon_audit
AFTER LOGON ON SCHEMA
BEGIN
  INSERT INTO hr_evnt_audit VALUES(
ora_sysevent,     sysdate,
    TO_CHAR(sysdate, 'hh24:mi:ss'),
    NULL,
    NULL
  );
  COMMIT;
END;
/
```

Schema Level Logoff System Event Trigger Step 1:
Logon to the database Step 2: Create a table.
Step 3: Write the trigger.

```
CREATE OR REPLACE TRIGGER log_off_audit
BEFORE LOGOFF ON SCHEMA
BEGIN
  INSERT INTO hr_evnt_audit VALUES(
    ora_sysevent,
    NULL,
    NULL,
    SYSDATE,
    TO_CHAR(sysdate, 'hh24:mi:ss')
  );
  COMMIT;
END;
/
```

Database Level System/Database Event Trigger
Example: How To Create Database Level Logoff event Trigger In Oracle PL/SQL

Step 1: Logon to the database

As only the user with ADMINISTER DATABASE TRIGGER system privilege can create a database level event trigger thus we need to make sure that this time we should log on to the database using one of these users. (SYSDBA)

Step 2: Create a Table

Step 3: Write the Database Level logoff system event trigger.

```
CREATE OR REPLACE TRIGGER db_lgof_audit
BEFORE LOGOFF ON DATABASE
BEGIN
  INSERT INTO db_evnt_audit
  VALUES(
    user,
    ora_sysevent,
    NULL,
    NULL,
    SYSDATE,
    TO_CHAR(sysdate, 'hh24:mi:ss')
    );
END;
/
```

Startup Trigger.

Startup triggers execute during the startup process of the database. In order to create a database event trigger for shutdown and startup events we either need to logon to the database as a user with DBA privileges such as sys or we must possess the ADMINISTER DATABASE TRIGGER system privilege.

Example

Step1: Logon to the database

In order to create a trigger on Startup Database Event first we will have to logon to our database using the user SYS with DBA privileges.

Step 2: Create a Table

To store the data generated by the execution of trigger we will require a table.

```
CREATE TABLE startup_audit
(
  Event_type  VARCHAR2(15),   event_date  DATE,
  event_time  VARCHAR2(15)
);
```

Step 3: Create the database Event Startup Trigger

```
CREATE OR REPLACE TRIGGER startup_audit
AFTER STARTUP ON DATABASE
BEGIN
  INSERT INTO startup_audit VALUES
(
    ora_sysevent,
    SYSDATE,
    TO_CHAR(sysdate, 'hh24:mm:ss')
  );
END;
/
```

Shutdown Triggers

SHUTDOWN triggers execute before database shutdown processing is performed. Similar to the startup trigger, only a user with DBA role or ADMINISTER DATABASE TRIGGER system privilege can create a shutdown trigger.

First 2 steps of creating a database event shutdown triggers are same as that of the startup trigger which we saw above.

```
CREATE OR REPLACE TRIGGER tr_shutdown_audit
BEFORE SHUTDOWN ON DATABASE
BEGIN
  INSERT INTO startup_audit VALUES(     ora_sysevent,
    SYSDATE,
    TO_CHAR(sysdate, 'hh24:mm:ss')
  );
END;
/
```

You can also use shutdown database event triggers for gathering your database system statistics.
Here is an example

```
CREATE OR REPLACE TRIGGER before_shutdown
BEFORE SHUTDOWN ON DATABASE
BEGIN
 gather_system_stats;
END;
/
```

Instead Of Trigger

Instead-of triggers in oracle database provide a way of modifying views that cannot be modified directly through the DML statements. By using Instead-of triggers, you can perform Insert, Update, Delete and Merge operations on a view in oracle database

Restriction on Instead-of View.

Instead-of triggers can control Insert, Delete, Update and Merge operations of the View, not the table. Yes you heard it right, you can write an instead-of trigger on Views only and not on tables in Oracle database. That is the

restriction that you have to comply with. Along with this you even have to comply with every general restriction that is imposed on all types of triggers

Uses of Instead-of trigger.
Since an Instead-of trigger can only be used with views therefore we can use them to make a nonupdatable view updatable as well as to override the default behavior of views that are updatable.

What are Modifiable and Non Modifiable Views?
A view is naturally modifiable if you do not require INSTEAD OF triggers to insert, delete or update data as well as if it complies to the restrictions discussed herewith. If the view query comprises of any of the mentioned constructs, then it is not naturally modifiable and therefore you cannot perform inserts, updates, or deletes on the view:

- Set operators
- Aggregate functions
- GROUP BY, CONNECT BY, or START WITH clauses
- The DISTINCT operator
- Joins (however, some join views are updatable)

In case a view consists of pseudo columns or expressions, then it is only possible to update it with an UPDATE statement and that also when it does not refer to any such pseudo columns or for that matter, expressions.

Syntax of Instead-Of Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
INSTEAD OF operation
ON view_name
FOR EACH ROW
BEGIN
        ---Your SQL Code— END;
/
```

Examples
Instead-of Insert Trigger
Instead-of trigger can be best demonstrated using a View joining two or more tables.
Step1: Create Tables
Table 1- trainer

CREATE TABLE trainer (full_name VARCHAR2(20));
Table 2- Subject

CREATE TABLE subject (subject_name VARCHAR2(15));
Insert dummy data into the above tables

INSERT INTO trainer VALUES (Sohail Ahmed);
INSERT INTO subject VALUES (Database Systems);

Step 2: Create View

```
CREATE VIEW db_lab_09_view AS
SELECT full_name, subject_name FROM trainer, subject;
```

This is a non-updatable view which you can confirm by executing any DML statement over it. Error as a result of DML operation on this view will be your confirmation.

Step 3: Create Trigger

```
CREATE OR REPLACE TRIGGER tr_Io_Insert
INSTEAD OF INSERT ON db_lab_09
FOR EACH ROW
BEGIN
  INSERT INTO trainer (full_name) VALUES (:new.full_name);
  INSERT INTO subject (subject_name) VALUES (:new.subject_name); END
```

Instead-Of Update Trigger

Instead-of update trigger will override the default behavior of your update operation when you execute the update statement and will let you update the data of the underlying tables over which your view is created.

Example:

Tables (Trainer and Subject) and View (db_lab_09) used in this example are the same as the ones we created.

```
CREATE OR REPLACE TRIGGER io_update
INSTEAD OF UPDATE ON db_lab_09
FOR EACH ROW
BEGIN
  UPDATE trainer SET FULL_NAME = :new.full_name
  WHERE FULL_NAME = :old.full_name;
  UPDATE subject SET subject_NAME = :new.subject_name
  WHERE subject_NAME = :old.subject_name;
END;
/
```

Instead-of Delete trigger Example.

Needless to say that executing DELETE DML on this view will return an error because of its non-updatable nature. Thus the only way to perform DELETE DML on this view is by using an Instead of trigger. Let's quickly create one.

```
CREATE OR REPLACE TRIGGER io_delete
INSTEAD OF DELETE ON vw_RebellionRider
FOR EACH ROW
BEGIN
  DELETE FROM trainer WHERE FULL_NAME = :old.FULL_NAME;
  DELETE FROM subject WHERE SUBJECT_NAME= :old.SUBJECT_NAME; END;
/
```

Lab Tasks:

DML Trigger Task:

- Create a DML trigger that logs changes to a specific table when records are inserted, updated, or deleted.
- Create a DML trigger that enforces a referential integrity constraint between two tables when inserting or updating records.
- Create a DML trigger that automatically updates a "last_modified" timestamp when a record is updated in a table.

DDL Trigger Tasks:

- Create a DDL trigger that logs all schema changes (e.g., table creations, modifications, and drops) in a dedicated audit table.
- Create a DDL trigger that prohibits any user from altering or dropping a specific critical table.
- Create a DDL trigger that prevents the creation of new tables with a specific naming pattern.

System/Database Trigger Task:

- Create a system trigger that captures information when a user logs in.
- Create a system trigger that sends an email notification to the DBA whenever a user with specific privileges logs in.
- Create a system trigger that automatically sets the session time zone for all users upon login.

Instead of Trigger Task:

- Create an "Instead of" trigger for a view that allows inserting data into multiple tables when an insert operation is performed on the view
- Create an "Instead of" trigger for a view that allows users to update records in a way that calculates and updates a computed column.
- Create an "Instead of" trigger for a view that allows users to insert records into multiple related tables through a single view.