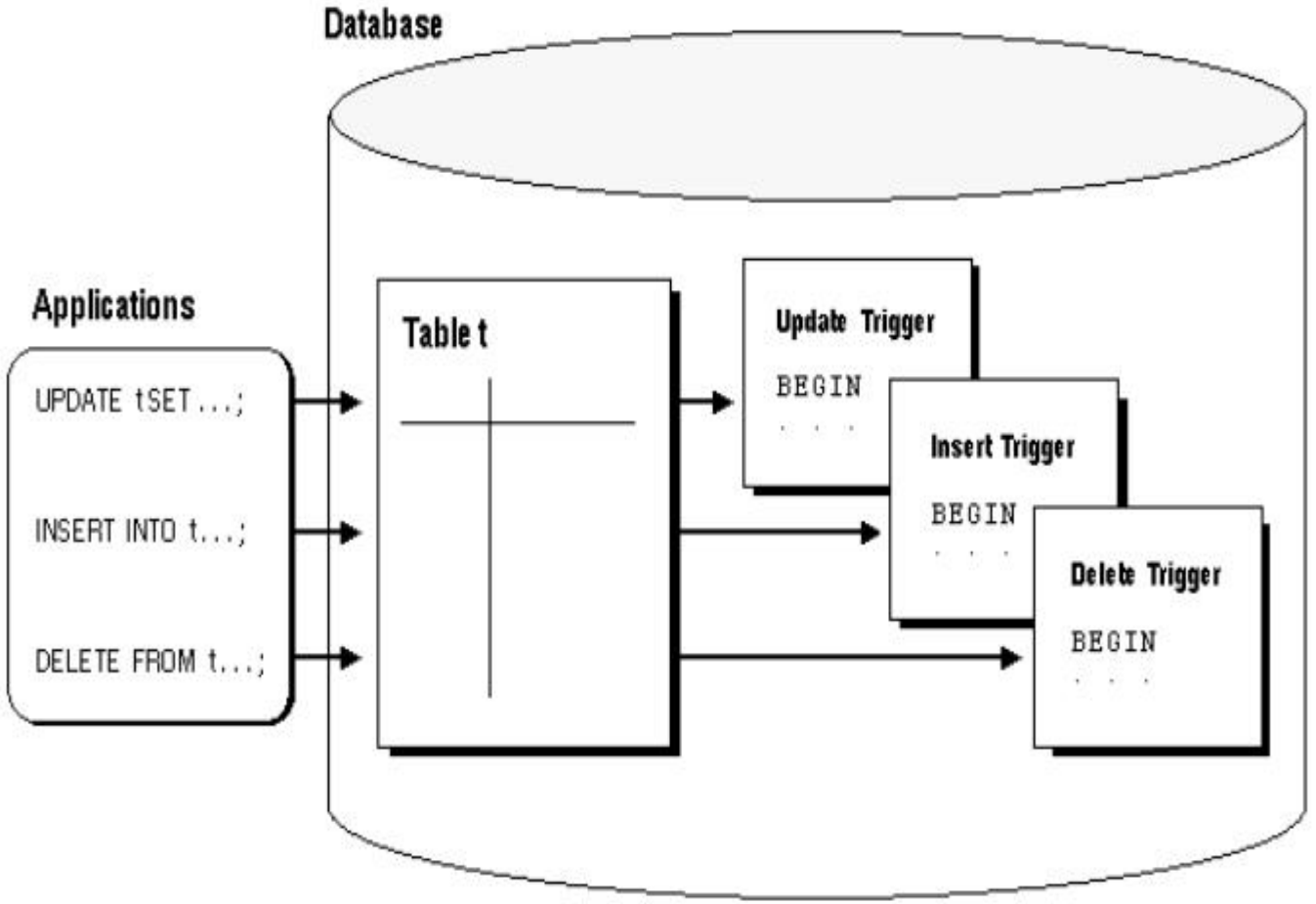
DATABASE TRIGGERS

- Database trigger – a stored PL/SQL program unit that is associated with a specific database table, or with certain view types and can be fired automatically in response to any DML events or a system event such as database startup.
- Two sections:
  - A named database event
  - A PL/SQL block that will execute when the event occurs

Database Trigger

Triggers get executed (fire) automatically when specified SQL DML operations – INSERT, UPDATE, or DELETE affecting one or more rows of a table.



USES-DATABASE TRIGGERS

- Database triggers can be used to perform any of the following tasks:
  - Audit data modification.
  - Log events transparently.
  - Enforce complex business rules.
    - Prevent DML operations on a table after regular business hours
  - Derive column values automatically.
  - Implement complex security authorizations.
  - Maintain replicate tables.
  - Gather statistics on table access
  - Publish information about events for a publish-subscribe environment such as that associated with web programming.

Difference between Trigger and Constraints

- Trigger affects only those rows, which are added after it is enabled.

- Constraints affects all the rows i.e. Validates the even already existing data before defining the constraint.

- Triggers:
  - are named PL/SQL blocks with declarative, executable, and exception handling sections.
  - are stand-alone database objects
  - do not accept arguments.

- To create/test a trigger, you (not the 'system' user of the trigger) must have appropriate access to all objects referenced by a trigger action.

- **Example:** To create a BEFORE INSERT trigger for the *employee* table requires you to have INSERT ROW privileges for the table.

Create Trigger Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER|INSTEAD OF} triggering_event ON
  {table_name | view_name} [referencing_clause]
[WHEN condition] [FOR EACH ROW]

[DECLARE
    Declaration statements]
BEGIN
    Executable statements
[EXCEPTION
    Exception-handling statements]
END;
```

The trigger body must have at least the executable section.
The declarative and exception handling sections are optional.
When there is a declarative section, the trigger body must start with the DECLARE keyword.
The WHEN clause specifies the condition under which a trigger should fire.

Trigger Types

- **BEFORE** and **AFTER** Triggers – trigger fires before or after the triggering event. Applies only to tables.
- **INSTEAD OF** Trigger – trigger fires instead of the triggering event. Applies only to views.
- **Triggering event** – a DML statement issued against the table or view named in the ON clause – example: INSERT, UPDATE, or DELETE.

DML triggers are fired by DML statements and are referred to sometimes as **row triggers**.

- **FOR EACH ROW** clause – a ROW trigger that fires once for each modified row.
- **STATEMENT** trigger – fires **once** for the DML statement.
- **Referencing_clause** – enables writing code to refer to the data in the row currently being modified by a different name.

Conditional Predicates for Detecting Triggering DML Statement

Conditional Predicate	TRUE if and only if:
INSERTING	An INSERT statement fired the trigger.
UPDATING	An UPDATE statement fired the trigger.
UPDATING ('*column*')	An UPDATE statement that affected the specified column fired the trigger.
DELETING	A DELETE statement fired the trigger.

```
SET SERVEROUTPUT On
CREATE OR REPLACE TRIGGER trig1
  BEFORE   INSERT OR  UPDATE OF sal, deptno OR
    DELETE   ON emp
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING('sal') THEN
      DBMS_OUTPUT.PUT_LINE('Updating salary');
    WHEN UPDATING('Deptno') THEN
      DBMS_OUTPUT.PUT_LINE('Updating department ID');
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Deleting');
  END CASE; END;
/
```

Example-1: A trigger program to display the trigger event that resulted into trigger execution
Save the file – **trg1_ex.sql**

SQL>@ trg1_ex.sql
Trigger created.
If errors
SQL> SHOW ERRORS TRIGGER trig1

SQL> insert into emp(empno, ename, sal, deptno)
values(119,'Akshay',3400,10);

Inserting

ROW Trigger – Accessing Rows

- Access data on the row currently being processed by using **two correlation identifiers** named **:old** and **:new**. These are special Oracle bind variables.
- The PL/SQL compiler treats the **:old** and **:new** records as records of type trigger_Table_Name%ROWTYPE.
- To reference a column in the triggering table, use the notation shown here where the *ColumnName* value is a valid column in the triggering table.

```
:new.ColumnName
:old.ColumnName
```

Bind Variables :old and :new Defined

DML Statement	:old	:new
INSERT	Undefined – all column values are NULL as there is no "old" version of the data row being inserted.	Stores the values that will be inserted into the new row for the table.
UPDATE	Stores the original values for the row being updated before the update takes place.	Stores the new values for the row – values the row will contain after the update takes place.
DELETE	Stores the original values for the row being deleted before the deletion takes place.	Undefined – all column values are NULL as there will not be a "new" version of the row being deleted.

Example

```
CREATE TABLE Emp (
        Empno NUMBER(4),
        Name  varchar2(10),
        salary NUMBER(7,2),
        Deptno     VARCHAR2(20));
```

This table contains employee information

```
CREATE TABLE Emp_log (
        Emp_id     NUMBER(4),
        Log_date   DATE,
        Old_salary NUMBER(7,2),
        New_salary NUMBER(7,2),
        Action     VARCHAR2(20),
        User_name varchar2(10));
```

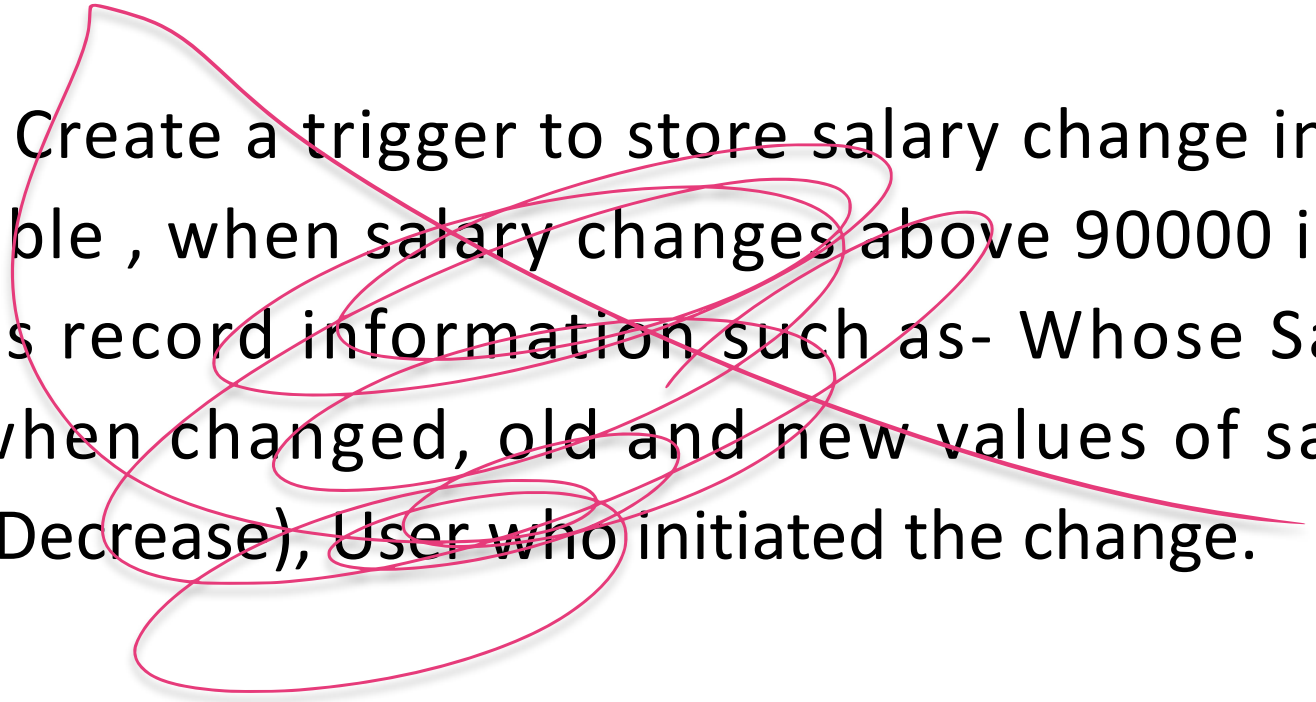
This table records the salary change events.

Example: Create a trigger to store salary change information into **EMP_log** table , when salary changes is made to **EMP** table. It has record information such as- Whose Salary has been changed, when changed, old and new values of salary, Action(I-increase, D-Decrease), User who initiated the change.

Example-2

```
CREATE OR REPLACE TRIGGER log_salary_increase
  AFTER UPDATE OF sal ON emp
  FOR EACH ROW
DECLARE
  user_action VARCHAR2(1);
BEGIN
  if(:new.sal>:old.sal) then
     user_action:='I';
  elsif :new.sal=:old.sal then
    user_action:='N';
  else
       user_action:='D';
   end if;
  INSERT INTO Emp_log
  VALUES (:NEW.empno,
  SYSDATE,:old.sal, :NEW.sal,user_action ,Use
  r);
  END;
```

WHEN clause- to specify condition under which Trigger has to fire



Example 3: Create a trigger to store salary change information into **EMP_log** table , when salary changes above 90000 is made to **EMP** table. It has record information such as- Whose Salary has been changed, when changed, old and new values of salary, Action(**I**-increase, **D**-Decrease), User who initiated the change.

Example-3

```
CREATE OR REPLACE TRIGGER log_salary_increase_when
  AFTER UPDATE OF sal ON emp1 WHEN (new.sal>90000)
  FOR EACH ROW
DECLARE
  user_action VARCHAR2(1);
BEGIN
  if(:new.sal>:old.sal) then
     user_action:='I';
  elsif :new.sal=:old.sal then
    user_action:='N';
else
     user_action:='D';
  end if;
INSERT INTO Emp_log1
VALUES (:NEW.empno,
SYSDATE,:old.sal, :NEW.sal,user_action ,U
r);
END;
```

STATEMENT Trigger - Example

Write a trigger to validate the salary updating action only during week-days

Note: Salary update statement may be updating multiple rows but Trigger is executed only once because It is System Trigger therefore FOR EACH ROW is not used

--User lab p: vm create another table EMP2;

```
create or replace trigger sal_update_weekDays before update of sal on emp2
begin
if to_char(sysdate,'DY') = 'SUN' then
   raise_application_error(-20111,'No changes can be made on Sunday.');
end if;
end;
/
```

STATEMENT Trigger - Example

```
/* CREATE TABLE users_log(User_name varchar2(10),Operation
varchar2(10),Login_Date Date);*/


CREATE OR REPLACE TRIGGER note_hr_logon_trigger

  AFTER LOGON  ON LAB.SCHEMA

BEGIN

  INSERT INTO users_log VALUES (USER, 'LOGON', SYSDATE);

END;

/
```

Using Correlation Variable- Trigger - Example

```
create table new (id number, new number);
        insert into new values(1,10);     insert into new values(2,30);
create or replace trigger trig_coRelVariable
after update on new referencing old as old_rowtyp new as new_rowtyp
for each row
begin
 if updating then
    DBMS_OUTPUT.PUT_LINE('OLD VALUE '||:old_rowtyp.new||' NEW
VALUE '||:new_rowtyp.new);
 else
  DBMS_OUTPUT.PUT_LINE(' SOme Error ...');
 end if;
end;
/
```

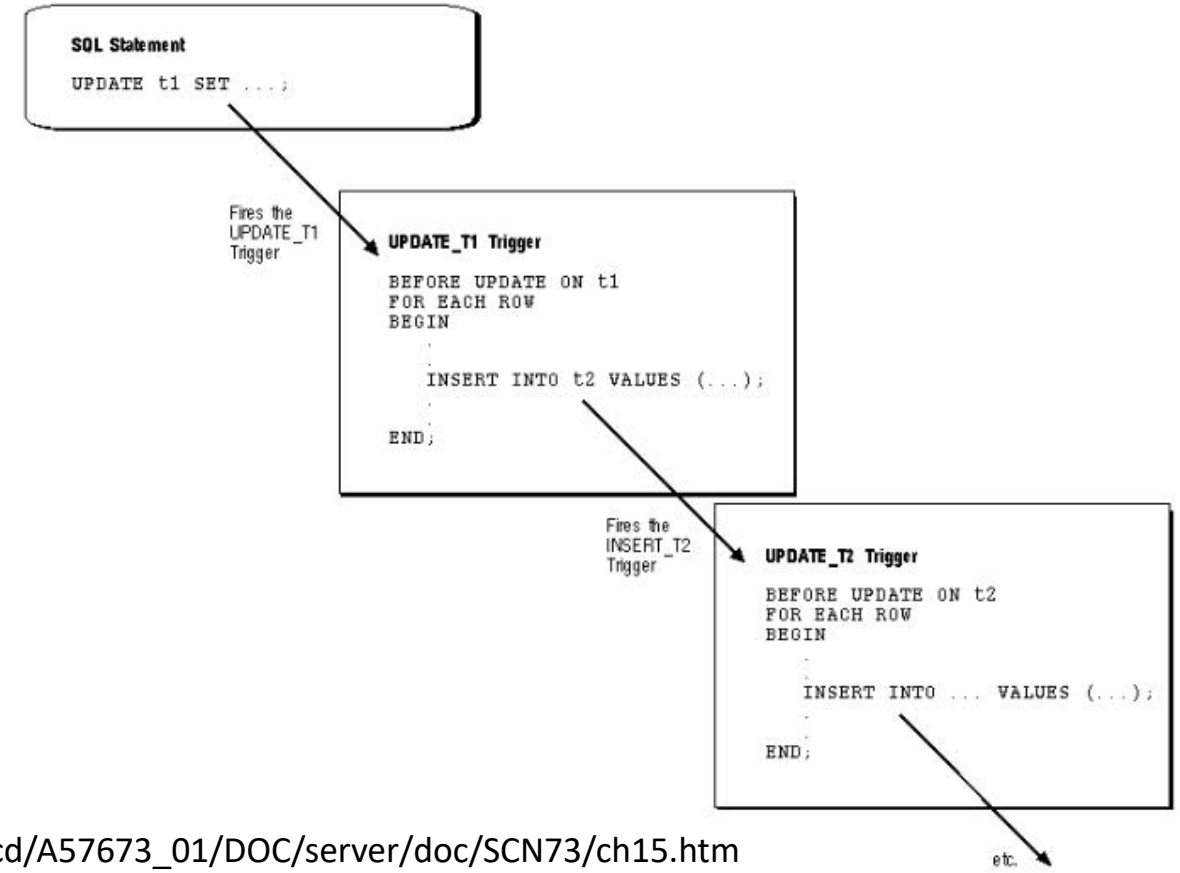
Dropping a Trigger

- The DROP TRIGGER statement drops a trigger from the database.
- If you drop a table, all associated table triggers are also dropped.
- The syntax is:

```
DROP TRIGGER trigger_name;
```

A Cautionary Note

When a trigger is fired, a SQL statement within its trigger action potentially can fire other triggers, . When a statement in a trigger body causes another trigger to be fired, the triggers are said to be ***cascading***.



https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch15.htm

. Cascading Triggers

Temporarily enabling/disabling trigger

To disable a trigger, you use the ALTER TRIGGER DISABLE statement

Syntax;

```
ALTER TRIGGER trigger_name DISABLE;
```

Example:

```
ALTER TRIGGER sal_update_weekDays_trg DISABLE;
```

To **disable all** triggers on a Table

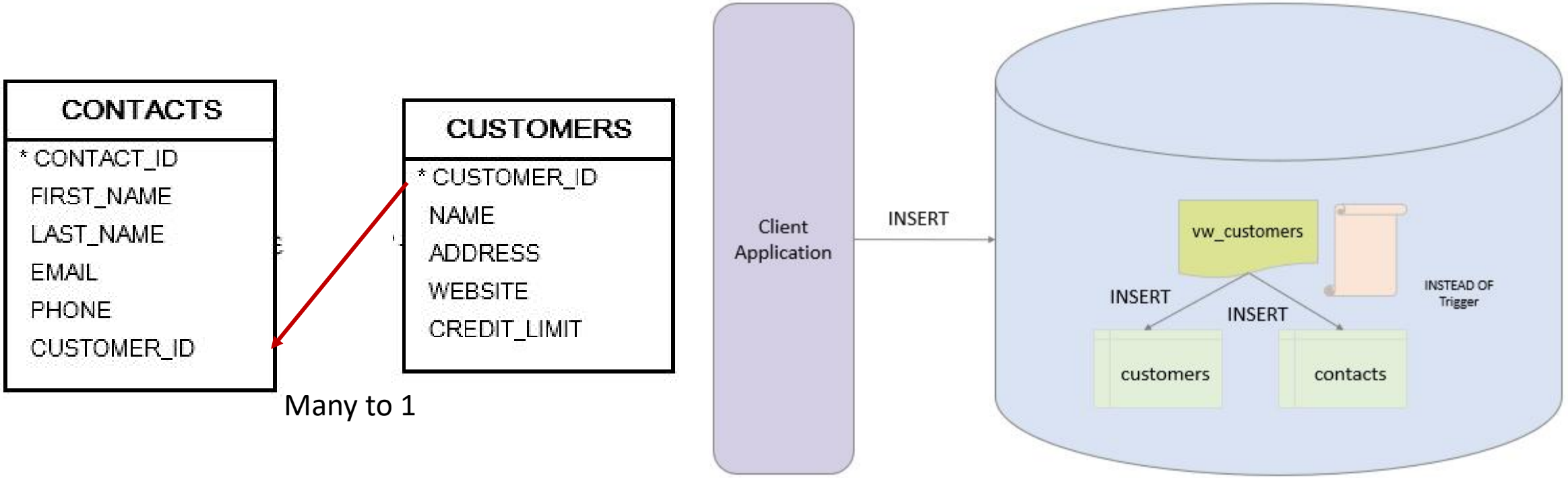
Syntax:

```
ALTER TABLE table_name DISABLE ALL TRIGGERS;
```

Example :

```
ALTER TABLE Emp DISABLE ALL TRIGGERS;
```

INSTEAD OF Trigger




```
CREATE VIEW vw_customers AS SELECT name, address, website,
credit_limit, first_name, last_name, email, phone FROM
customers Cust,contacts Cont where
Cust.Customer_id=Cont.Customer_id;
```

INSTEAD OF Trigger

Example: Create a view VW_emp3_dept3 based on Emp3(Empno, ename, sal, deptno) & Dept3(Deptno, Dname,Loc, Budget). Write a trigger to update base tables Dept3,Emp3 whenever VW_emp3_dept3 view is updated.

```
set serveroutput on;
CREATE OR REPLACE TRIGGER emp_dept_InsteadoF_trg
   INSTEAD OF INSERT ON VW_emp3_dept3   FOR EACH ROW
DECLARE
BEGIN
   DBMS_OUTPUT.PUT_LINE(' Entered deptno '||:new.deptno);
         insert into dept3 values(:new.deptno,:new.dname,null,null);
         insert into emp3 (empno,ename,deptno) values(:new.empno,:new.ename,:new.deptno);
end;
```

END