Example 1

Create a transparent system for maintenance of customer records by using trigger. If any UPDATE OR DELETE operations performed on the table CUST then id, date of operation, type of operation and username fields are stored in CUST_AUDIT table.

CUST		CUST_AUDIT	
Field name	Data type and width	Field name	Data type and width
Id	Number(5)	Id	Number(5)
Name	Varchar2(15)	O_Dt	Date
Age	Number(2)	User_name	Varchar2(15)
Address	Varchar2(10)	Operation	Varchar2(15)
Salary	Number(5)		

```
CREATE OR REPLACE TRIGGER AUDIT_CUST
AFTER UPDATE OR DELETE ON CUST
FOR EACH ROW
DECLARE
Oper varchar2(15);
BEGIN
    IF updating THEN
        Oper:='UPDATE';
    END IF;
    IF deleting THEN
        Oper:='DELETE';
    END IF;
    INSERT INTO CUST_AUDIT
    VALUES (:OLD.ID, SYSDATE, USER, Oper);
END;
```

Example 2

Write a database trigger, which will not allow any transaction after office hours (after 7:00 pm) and on Sunday.

```
CREATE OR REPLACE TRIGGER EMP_T1
BEFORE INSERT OR UPDATE OR DELETE
ON EMP
FOR EACH ROW
DECLARE
    dy varchar2(10);
    dt number;
BEGIN
  dy:= to_char(sysdate,'dy');
  dt:=to_number(to_char(sysdate,'hh24'));

  if  inserting and (dy='sun' or dt > 19)  then
       raise_application_error(-20001,'Cannot insert today');
  end if;

  if  updating and (dy='sun' or dt > 19) Then
     raise_application_error(-20002, 'Cannot update today');
  end if;

  if  deleting and (dy='sun' or dt > 19) Then
     raise_application_error(-20003, 'Cannot delete today');
  end if;

 END;
```

Example 3:

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters:

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

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:**

```
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(5);
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;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Trigger created.
```

Here following two points are important and should be noted carefully:

- 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.

- 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 DELETE operation on the table.

Triggering a Trigger

Let us 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 (7, 'Kriti', 22, 'HP', 7500.00 );
```

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

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

Because this is a new record so old salary is not available and above result is coming as null. Now, let us perform one more DML operation on the CUSTOMERS table. Here is one UPDATE statement which will update an existing record in the table:

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

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

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

Example 3:**Create automatic primary key using sequence.**

```
SQL> CREATE TABLE FAC_CUST
      (custno varchar2 (10), cust_name varchar2 (15),
cust_age number (5));

Table created.

SQL> create sequence cust_seq increment by 1 start with 1;

Sequence created.

SQL> create or replace trigger fac_cust_t1 before insert on fac_cust
 for each row
  2  declare
  3  pk varchar2(10);
  4  begin
  5  select 'C' || to_char (cust_seq.nextval) into pk from dual;
  6  :new.custno:=pk;
  7  end;
  8  /

Trigger created.
```

Example 4:**Create automatic primary key using max function.**

```
SQL> create or replace trigger fac_cust_t1
  2   before insert on fac_cust
  3   for each row
  4   declare
  5   maxno number (2);
  6   pk varchar2(10);
  7   begin
  8   select max (to_number (substr (custno,2))) into maxno from fac_cust;
  9  pk := 'C'||to_char(maxno+1);
 10  :new.custno:=pk;
 11  end;
 12  /
Trigger created.
```

Example 5: **Create automatic primary key using LOOKUP table.**

```
CREATE OR REPLACE TRIGGER fac_cust_t1
BEFORE INSERT ON fac_cust
FOR EACH ROW

DECLARE
    lpk varchar2(10);
BEGIN
    BEGIN
        select pk_value into lpk from pkey_lookup;
   EXCEPTION
        when no_data_found then
            lpk := 'C1';
   END;
  :new.custno := lpk;
  lpk := 'C' || to_char(to_number(substr(lpk,2) + 1));
  IF lpk = 'C2' THEN
        INSERT INTO pkey_lookup VALUES (lpk);
  ELSE
        UPDATE pkey_lookup SET pk_value = lpk;
  END IF;
END;
```

EXAMPLE 6:

Write a database trigger that not allowing the change in emp table after Business hours (From 10:00 AM to 5.00 PM) from Monday to Saturday There is no restriction on viewing the data .

```
----------------------------------------
create or replace trigger timer_03b020
   before insert or update or delete
   on emp20
begin
   if deleting then
                raise_application_error(-20000.'Delete restriction.');
   end if;
   if inserting then
               raise_application_error(-20001,'Insert restricted.');
```

```
   end if;
  end if;
   if updating then
    if to_char(sysdate,'dy')<> 'mon' or
                to_char(sysdate,'dy')<> 'tue' or
                to_char(sysdate,'dy')<> 'wed' or
                to_char(sysdate,'dy')<> 'thu' or
                to_char(sysdate,'dy')<> 'fri' or
                to_char(sysdate,'dy')<> 'sat' or
           to_number(to_char(sysdate,'hh24')) < 10 or
                to_number(to_char(sysdate,'hh24')) > 17 then
            raise_application_error(-20002,'Updation restricted');
    end if;
  end if;
end;
/
```

- **TO FIND ERRORS IN PL/SQL :**

Select * from user_errors where name = 'TABLE_NAME' (capital letter)

- **TO FIND SOURCE IN PL/SQL :**

Select * from user_source where name = 'FUNCTION_NAME' (capital letter)