INDEX:

Experiment 10

Aim: Create of insert trigger, delete trigger and update trigger.

1. To write a TRIGGER to ensure that Bus table does not contain duplicate of null values in Bus_No column.

A Trigger is a named database object which defines some action that the database should take when some databases related event occurs. Triggers are executed when you issues a data manipulation command like INSERT, DELETE, UPDATE on a table for which the trigger has been created. They are automatically executed and also transparent to the user. But for creating the trigger the user must have the CREATE TRIGGER privilege. In this section we will describe you about the syntax to create and drop the triggers and describe you some examples of how to use them.

CREATE TRIGGER

The general syntax of CREATE TRIGGER is :

```
CREATE TRIGGER trigger_name trigger_time trigger_event ON tbl_name FOR
EACH ROW trigger_statement
```

By using above statement we can create the new trigger. The trigger can associate only with the table name and that must be refer to a permanent table. Trigger_time means trigger action time. It can be BEFORE or AFTER. It is used to define that the trigger fires before or after the statement that executed it. Trigger_event specifies the statement that executes the trigger. The trigger_event can be any of the DML Statement : INSERT, UPDATE, DELETE.

We can not have the two trigger for a given table, which have the same trigger action time and event. For Instance : we cannot have two BEFORE INSERT triggers for same table. But we can have a BEFORE INSERT and BEFORE UPDATE trigger for a same table.

Trigger_statement have the statement that executes when the trigger fires but if you want to execute multiple statement the you have to use the BEGIN…END compound statement.

We can refer the columns of the table that associated with trigger by using the OLD and NEWkeyword. OLD.column_name is used to refer the column of an existing row before it is deleted or updated and NEW.column_name is used to refer the column of a new row that is inserted or after updated existing row.

In INSERT trigger we can use only NEW.column_name because there is no old row and in a DELETE trigger we can use only OLD.column_name because there is no new row. But in UPDATE trigger we can use both, OLD.column_name is used to refer the columns of a row before it is updated and NEW.Column_name is used to refer the column of the row after it is updated.

a) CREATE OR RELPLACE TRIGGER trig1 before insert on Bus for each row

```
DECLARE a number;
BEGIN
        if(:new.Bus_No is Null) then
                raise_application_error(-20001,'error:: Bus_No cannot be null');
                else
                select count(*) into a from Bus where Bus_No =:new. Bus_No;
                if(a=1) then
                    raise_application_error(-20002,'error:: cannot have duplicate Bus_No ');
                end if;
        end if;
END;
```

RESULT:

```
SQL> @trigger
Trigger created.
SQL> select * from Bus;
BUS_NO     SOURCE          DESTINATION
-----          -------------------- --------------------
110                 hyd            ban
221                 hyd            chn
412                 hyd            mum
501                 hyd            kol
SQL> insert into Bus values(&Bus_No,'&source','&destination');
Enter value for Bus_No: null
Enter value for source: Chen
Enter value for destination: hyd
```

```
old 1: insert into Bus values(&Bus_No, '&source', '&destination')
new 1: insert into Bus values(null,Chen','hyd')
insert into Bus values(null,’Chen','hyd')
*
SQL> /
Enter value for Bus_No: 110
Enter value for source:KOL
Enter value for destination: hyd
old 1: insert into Bus values(&Bus_No, '&source', '&destination')
new 1: insert into Bus values(110,KOL','hyd')
insert into Bus values(110,’KOL','hyd')
*
```

b) Create Trigger updchek before update on Ticket

```
For Each Row
Begin
        If New.Ticket_No>60 Then
                Set New.Ticket_No=Ticket_No;
        Else
                Set New.Ticket_No=0;
        End If
End.
SQL> @trigger
Trigger created.
```

c) CREATE OR RELPLACE TRIGGER trig1 before insert on Passenger for each row

```
DECLARE a number;
BEGIN
        if(:new.PNR_NO is Null) then
                raise_application_error(-20001,'error:: PNR_NO cannot be null');
                else
                select count(*) into a from Passenger where PNR_NO =:new. PNR_NO;
                if(a=1) then
                    raise_application_error(-20002,'error:: cannot have duplicate PNR_NO ');
                end if;
        end if;
END;
SQL> @trigger
Trigger created.
```

Experiment 11

Aim: Creation of stored procedures, Execution of procedure and modification of procedure.

A stored procedure is a procedure (like a subprogram in a regular computing language) that is stored (in the database). Correctly speaking, MySQL supports "routines" and there are two kinds of routines: stored procedures which you call, or functions whose return values you use in other SQL statements the same way that you use pre-installed MySQL functions like pi(). I'll use the word "stored procedures" more frequently than "routines" because it's what we've used in the past, and what people expect us to use.

```
Create procedure myproc()
BEGIN
     SELECT COUNT(Tickets) from ticket where age>=20;
END;
PL/SOL procedure successfully completed
Create procedure myproc(in_customer_id INT)
BEGIN
    DECLARE V_id INT;
    DECLARE V_name VARCHAR(30);
    DECLARE c1 CURSOR FOR SELECT stdid,stdfirstname FROM students WHERE
stdid=in_customer_id;
OPEN c1;
       FETCH c1 into V_id,V_name;
CLOSE c1;
END;
/
PL/SQL procedure successfully completed.
```

Experiment 12

Aim: To write a Cursor to display the list of Male and Female Passengers.

Cursors are used when the SQL Select statement is expected to return more than one row. Cursors are supported inside procedures and functions. Cursors must be declared and its definition contains the query. The cursor must be defined in the DECLARE section of the program. A cursor must be opened before processing and close after processing.

Syntax to declare the cursor:

DECLARE <cursor_name> CURSOR FOR <select_statement>

Multiple cursors can be declared in the procedures and functions but each cursor must have a unique name. And in defining the cursor the select_statement cannot have INTO clause.

Syntax to open the cursor : OPEN <cursor_name>

By this statement we can open the previously declared cursor.

Syntax to store data in the cursor :

FETCH <cursor_name> INTO <var1>,<var2>…….

The above statement is used to fetch the next row if a row exists by using the defined open cursor.

Syntax to close the cursor : CLOSE <cursor_name>

By this statement we can close the previously opened cursor. If it is not closed explicitly then a cursor is closed at the end of compound statement in which that was declared.

```
DECLARE
cursor c(jb varchar2) is select Name from Passenger where Sex=m;
pr Passenger.Sex%type;
BEGIN
open c('m');
dbms_RESULT.put_line(' Name of Male Passenger are:');
```

```
loop
fetch c into pr;
exit when c%notfound;
dbms_RESULT.put_line(pr);
end loop;
close c;
open c('f');
dbms_RESULT.put_line(' Name of female Passengers are:');
loop
fetch c into em;
exit when c%notfound;
dbms_RESULT.put_line(em);
end loop;
close c;
END;
```

RESULT:

```
Name of Male Passenger are:
SACHIN
rahul
rafi
salim
riyaz
Name of female Passengers are:
swetha
neha
PL/SQL procedure successfully completed.
```

b) To write a Cursor to display List of Passengers from Passenger Table.

```
DECLARE
cursor c is select PNR_NO, Name, Age, Sex from Passenger ;
i Passenger.PNR_NO%type;
```

```
j Passenger.Name%type;
k Passenger.Age%type;
l Passenger.Sex%type;
BEGIN
open c;
dbms_RESULT.put_line('PNR_NO, Name, Age, Sex of Passengers are:= ');
loop
fetch c into i, j, k, l;
exit when c%notfound;
dbms_RESULT.put_line(i||' '||j||' '||k||' '||l);
end loop;
close c;
END;
```

RESULT:

```
SQL>@Passenger
      PNR_NO     NAME                 AGE SEX
   ----------     -------------------- ---------- ----------
          1          SACHIN               12    m
          2          rahul                43    m
          3          swetha               24    f
          4          rafi                 22    m
PL/SQL procedure successfully completed.
```