Lab 9
Trigger, Transaction Management, Date
Version 2.0

Trigger

A trigger is a PL/SQL block that executes when a particular event occurs.
A PL/SQL trigger has two forms:

- Application triggers, which are created as blocks of code that execute when a nominated application event occurs in an Oracle Forms execution environment

Database triggers, which are named blocks of code that are associated and execute when a nominated database or table event occurs.

Run the script create2006.sql. Then create the following trigger with a separate script.

```
CREATE OR REPLACE TRIGGER check_salary BEFORE INSERT OR UPDATE ON
employees
FOR EACH ROW
DECLARE
   c_min constant number(8,2) := 1000.0;
   c_max constant number(8,2) := 500000.0;
BEGIN
  IF :new.salary > c_max OR :new.salary < c_min THEN
  RAISE_APPLICATION_ERROR(-20000,'New salary is too small or large');
END IF;
END;
/
```

The check_salary trigger example shows a database trigger that executes before either an INSERT or an UPDATE operation occurs on the EMPLOYEES table. The trigger code checks whether the salary column value is within an acceptable range. If the value is outside the specified range, then the code uses the RAISE_APPLICATION_ERROR built-in procedure to fail the operation. The: new syntax, which is used in the example, is a special bind/host variable that can be used in row-level triggers.

Now, consider the following example. Say, there would be a table called student having the following columns and data types.

```
roll_no number(4) PRIMARY KEY
name varchar2(50) NOT NULL
marks number(3)
grade char(1)
course_id number(4) FOREIGN KEY
```

Now, we will create a trigger on student table so that when we insert or update marks field the grade field should automatically get updated or filled.

```
marks>80 grade =A
marks 80-70 grade = B
marks 70-60 grade =C
marks 60- 50 grade = D
marks 50-40 grade = E
marks<=40 grade = F
```

First, we will create the table as instructed.

```
CREATE TABLE STUDENT
(
roll_no number(4) PRIMARY KEY,
name varchar2(50) NOT NULL,
marks number(3),
grade char(1),
course_id number(4)
);
```

Now, we will create the trigger as instructed.

```
CREATE TRIGGER TR_GRADE
BEFORE UPDATE OR INSERT ON STUDENT
FOR EACH ROW
BEGIN
IF :NEW.MARKS>80 THEN
:NEW.GRADE:='A';
ELSIF :NEW.MARKS>70 AND :NEW.MARKS<80 THEN
:NEW.GRADE:='B';
ELSIF :NEW.MARKS>60 AND :NEW.MARKS<70 THEN
:NEW.GRADE:='C';
ELSIF :NEW.MARKS>50 AND :NEW.MARKS<60 THEN
:NEW.GRADE:='D';
ELSIF :NEW.MARKS>40 AND :NEW.MARKS<50 THEN
:NEW.GRADE:='E';
ELSIF :NEW.MARKS<=40 THEN
:NEW.GRADE:='F';
END IF;
END TR_GRADE;
/
```

Now, insert a dummy value like the following and see what happens. The trigger automatically calculates the grade and inserts it.

```
insert into student VALUES (1,'SATHISH',3,NULL,123);
SELECT * FROM STUDENT;
```

Transaction Management

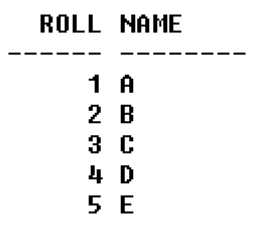
Transactions begin with a single DML statement and end (successfully or unsuccessfully) when one of the following events occurs:

- Either a COMMIT or ROLLBACK statement is executed. A COMMIT statement makes the changes to the table permanent, while the ROLLBACK undoes the changes to the table.
- The user exits SQL*Plus or iSQL*Plus normally (automatic COMMIT).
- A DDL (Data Definition Language) or DCL (Data Control Language) statement is executed (automatic COMMIT).
- The database crashes (automatic ROLLBACK).
- The SQL*Plus or iSQL*Plus session crashes (automatic ROLLBACK).

Write down the following script to create a table called TEST and to insert dummy values into it (Script 1.sql).

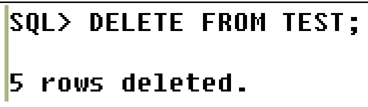
```
DROP TABLE test;
CREATE TABLE test (
      Roll  int,
      Name  Varchar (10)
);

INSERT INTO test VALUES (1, 'A');
INSERT INTO test VALUES (2, 'B');
INSERT INTO test VALUES (3, 'C');
INSERT INTO test VALUES (4, 'D');
INSERT INTO test VALUES (5, 'E');

COMMIT;
```

ON sqlplus, run the command- SELECT * FROM test. This will show your table.



```
  ROLL NAME
------ --------
     1 A
     2 B
     3 C
     4 D
     5 E
```

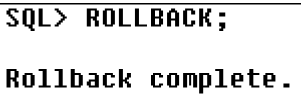
Now, Delete everything from this table with DELETE FROM test command.



```
SQL> DELETE FROM TEST;

5 rows deleted.
```

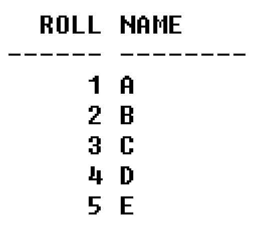
Say, now, you realize you did not use a WHERE clause in this command, so everything will be deleted. What do you do?

Simply write ROLLBACK on sqlplus.



```
SQL> ROLLBACK;

Rollback complete.
```

This command will take you to the last consistency state of the database. To see the effect, run the command SELECT * FROM test. The DELETE command you have just entered has been rolled back.



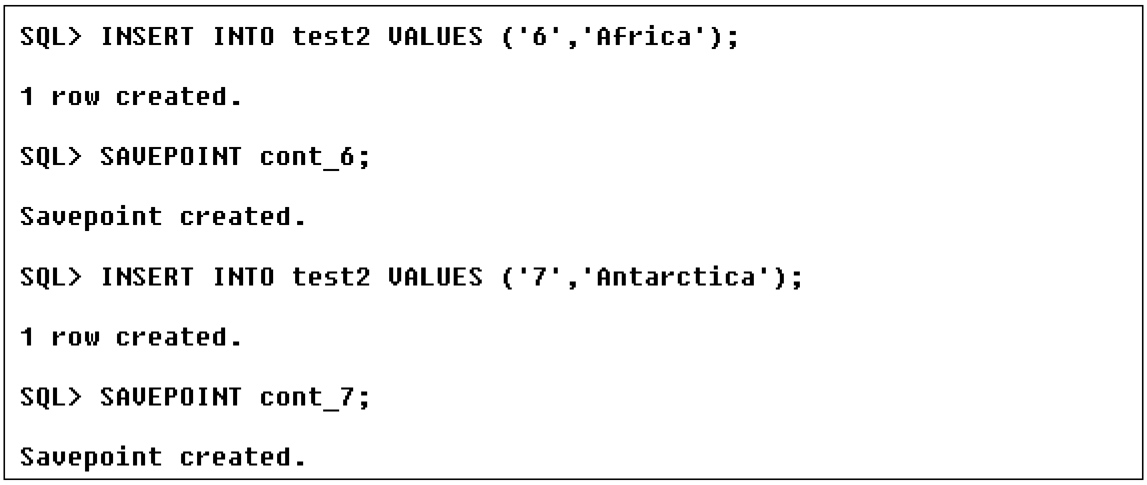
```
  ROLL NAME
------ --------
     1 A
     2 B
     3 C
     4 D
     5 E
```

Now, Write down the following script and run it from the sqlplus (Script 2.sql).

```
DROP TABLE test2;
```

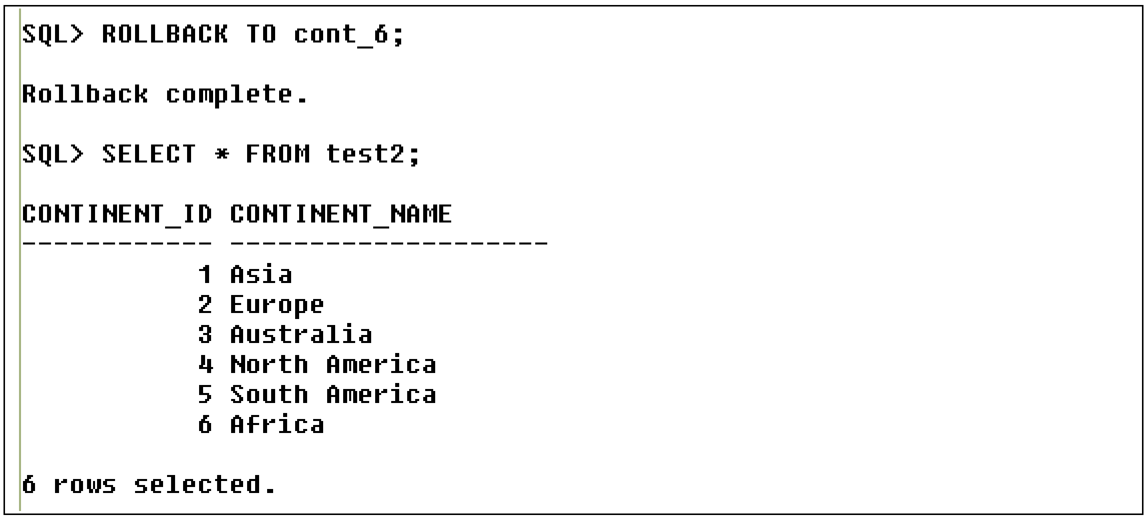
```
CREATE TABLE test2 (
      Continent_ID     int,
      Continent_Name   varchar(20)
);

INSERT INTO test2 VALUES (1, 'Asia');
INSERT INTO test2 VALUES (2, 'Europe');
INSERT INTO test2 VALUES (3, 'Australia');
INSERT INTO test2 VALUES (4, 'North America');
INSERT INTO test2 VALUES (5, 'South America');
commit;
```

We have now created a table with 5 IDs and Names of 5 Continents. We still have two continents left. We will insert them from sqlplus and for each of them, we will create a SAVEPOINT.



```
SQL> INSERT INTO test2 VALUES ('6','Africa');

1 row created.

SQL> SAVEPOINT cont_6;

Savepoint created.

SQL> INSERT INTO test2 VALUES ('7','Antarctica');

1 row created.

SQL> SAVEPOINT cont_7;

Savepoint created.
```

Now, we will jump into the SAVEPOINT cont_6. It means that we have saved a point after inserting Africa, & now we are rolling back there. Therefore, every other DML executed after that point will be ignored.

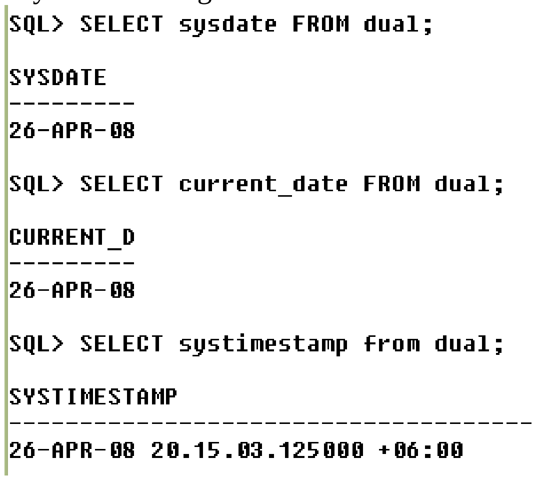


```
SQL> ROLLBACK TO cont_6;

Rollback complete.

SQL> SELECT * FROM test2;

CONTINENT_ID CONTINENT_NAME
------------ --------------------
           1 Asia
           2 Europe
           3 Australia
           4 North America
           5 South America
           6 Africa

6 rows selected.
```

Date

We will take a look at the functions Oracle offers to display the current date. And of course, we are dealing with DUAL table.

Try the followings-



```
SQL> SELECT sysdate FROM dual;

SYSDATE
---------
26-APR-08

SQL> SELECT current_date FROM dual;

CURRENT_D
---------
26-APR-08

SQL> SELECT systimestamp from dual;

SYSTIMESTAMP
-------------------------------------
26-APR-08 20.15.03.125000 +06:00
```

We can also make some arithmetic operations on date fields. To see this, run the script 5.sql. The table joining contains employee name, their viva and joining date. Now, run the following command-

```
SELECT Employee, Viva_date, Joining_date
FROM joining
WHERE Viva_date - Joining_date !=0;
```

This command will show those employees who are not so lucky- they have joined after some days from their viva. ☺

We can add months to a record in a specific column with ADD_MONTHS function. For example, run the following command-

```
SELECT ADD_MONTHS (Joining_date,6) AS Six_months_Extension
FROM joining
WHERE Employee = 'A';
```

This will add 6 months to the joining date of employee 'A'.

To subtract month, we also use the same function but only we change the month with a negative sign. Try the following-

```
SELECT ADD_MONTHS (Joining_date, -6) AS Six_months_Extension
FROM joining
WHERE Employee = 'A';
```

This will subtract 6 months from the joining date of employee 'A'.

The function GREATEST is used to find out the latest dates and LEAST is used to find out the earliest dates. For example, try the following-

```
SELECT LEAST (TO_DATE ('22-DEC-2007'),TO_DATE ('12-DEC-2008'))
FROM dual;
SELECT  GREATEST  (TO_DATE  ('22-DEC-2007'),TO_DATE  ('12-DEC-
2008'))
FROM dual;
```

Remember, you will have to use TO_DATE function to convert the date into date (!!) because LEAST and GREATEST are the only Oracle date functions that treats its arguments as STRINGS.

The function LAST_DAY, if a DATE column is given as the argument, this function returns the last day of the month mentioned in that record. For example, try this-

```
SELECT LAST_DAY (Viva_date)
FROM joining;
```

EXTRACT is a handy function. It gives you either Day, Month or Year of a particular DATE column. For example-

```
SELECT employee, EXTRACT(Year FROM Viva_date) AS Year
FROM joining;
```

Will return 2002 as you have extracted only the year from Viva_date.

Similarly, try this-

```
SELECT employee, EXTRACT(Month FROM Viva_date) AS Month
FROM joining;
```