Ayush Goyal

190905522 CSE D 62

DBS Lab-8 (Week 8) – Procedures, Functions & Triggers

Procedures:

1. **Based on the University Database Schema in Lab 2, write a procedure which takes the dept_name as input parameter and lists all the instructors associated with the department as well as list all the courses offered by the department. Also, write an anonymous block with the procedure call.**

CODE:

```
CREATE OR REPLACE PROCEDURE listInst (deptName Instructor.dept_name%type) IS
CURSOR curseInst (deptName Instructor.dept_name%type) IS
SELECT name FROM Instructor WHERE dept_name = deptName;
CURSOR curseCourses (deptName Instructor.dept_name%type) IS
SELECT course_id FROM Course WHERE dept_name = deptName;
BEGIN
    dbms_output.put_line('.........................');
    dbms_output.put_line('-- DEPARTMENTs INSTRUCTORS --');
    FOR row IN curseInst (deptName)
    LOOP
        dbms_output.put_line(' '||row.name);
    END LOOP;
    dbms_output.put_line('.........................');
    dbms_output.put_line('-- COURSES --');
    FOR row IN curseCourses (deptName) LOOP
        dbms_output.put_line(' ' || row.course_id);
    END LOOP;
END;
/

DECLARE
BEGIN
      listInst('Comp. Sci.');
END;
/
```

OUTPUT:

```
SQL> DECLARE
  2  BEGIN
  3  listInst('Physics');
  4  END;
  5  /
.........................
-- DEPARTMENTs INSTRUCTORS --
Einstein
Gold
.........................
-- COURSES --
PHY-101
```

```
Procedure created.

SQL> DECLARE
  2  BEGIN
  3  listInst('Comp. Sci.');
  4  END;
  5  /
.........................
-- DEPARTMENTs INSTRUCTORS --
Srinivasan
Katz
Brandt
.........................
-- COURSES --
CS-101
CS-190
CS-315
CS-319
CS-347

PL/SQL procedure successfully completed.

SQL>
```

2. **Based on the University Database Schema in Lab 2, write a Pl/Sql block of code that lists the most popular course (highest number of students take it) for each of the departments. It should make use of a procedure course_popular which finds the most popular course in the given department.**

CODE:

```
CREATE OR REPLACE PROCEDURE course_popular IS
CURSOR cursepop IS
WITH studentenroll as (select course_id,count(distinct ID) as student_count fr
om takes group by course_id),
studenmod as (select course_id,student_count,dept_name from studentenroll natu
ral join course),
deptmax as (select max(student_count) as dept_high,dept_name from course natur
al join studenmod group by dept_name)
select dept_high,course_id,dept_name from studenmod natural join deptmax where
 student_count=dept_high;
BEGIN
    FOR row IN cursepop LOOP
        dbms_output.put_line('Department name : '||row.dept_name);
        dbms_output.put_line(' Course ID : ' || row.course_id);
        dbms_output.put_line('Number of student enrolled : '||row.dept_high);
        dbms_output.put_line('-----------------------------------------------
---');
    END LOOP;
END;
/

DECLARE
BEGIN
    dbms_output.put_line('----- ALL DEPARTMENTS HIGHEST ENROLLED COURSES -----
-');
```

```
    course_popular;
END;
/
```

OUTPUT:

```
Procedure created.

SQL> DECLARE
  2  BEGIN
  3      dbms_output.put_line('----- ALL DEPARTMENTS HIGHEST ENROLLED COURSES ------');
  4      course_popular;
  5  END;
  6  /
----- ALL DEPARTMENTS HIGHEST ENROLLED COURSES ------
Department name : Biology
Course ID : BIO-101
Number of student enrolled : 1
-------------------------------------------------
Department name : Biology
Course ID : BIO-301
Number of student enrolled : 1
-------------------------------------------------
Department name : Comp. Sci.
Course ID : CS-101
Number of student enrolled : 6
-------------------------------------------------
Department name : Elec. Eng.
Course ID : EE-181
Number of student enrolled : 1
-------------------------------------------------
Department name : Finance
Course ID : FIN-201
Number of student enrolled : 1
-------------------------------------------------
Department name : History
Course ID : HIS-351
Number of student enrolled : 1
-------------------------------------------------
Department name : Music
Course ID : MU-199
Number of student enrolled : 1
-------------------------------------------------
Department name : Physics
Course ID : PHY-101
Number of student enrolled : 1
-------------------------------------------------

PL/SQL procedure successfully completed.

SQL>
```

Functions:

3. **Write a function to return the Square of a given number and call it from an anonymous block.**

CODE:

```
CREATE OR REPLACE FUNCTION square (x number)
RETURN number AS s number;
BEGIN
    s := x * x;
    RETURN s;
END;
/
```

```
DECLARE
BEGIN
    dbms_output.put_line('5 ^ 2 = '||square(5));
END;
/
```

OUTPUT:

```
SQL> CREATE OR REPLACE FUNCTION square (x number)
  2  RETURN number AS s number;
  3  BEGIN
  4      s := x * x;
  5      RETURN s;
  6  END;
  7  /

Function created.

SQL>
SQL> DECLARE
  2  BEGIN
  3      dbms_output.put_line('5 ^ 2 = '||square(5));
  4  END;
  5  /
5 ^ 2 = 25

PL/SQL procedure successfully completed.

SQL>
```

4. **Based on the University Database Schema in Lab 2, write a Pl/Sql block of code that lists the highest paid Instructor in each of the Department. It should make use of a function department_highest which returns the highest paid Instructor for the given branch.**

CODE:

```
CREATE OR REPLACE FUNCTION department_highest (dName Department.dept_name%type
)
RETURN Instructor.salary%type as
pop Instructor.salary%type;
BEGIN
    select max(salary) into pop
    from Instructor group by Instructor.dept_name having Instructor.dept_name
in (select dept_name

    from Instructor

    where dept_name = dName);
    return pop;
END;
/

DECLARE
    maxs Instructor.salary%type;
```

```
    cursor c1 is select distinct dept_name from department;
BEGIN
    for dn in c1 loop
        maxs := department_highest(dn.dept_name);
        dbms_output.put_line('Highest paid salary in '||dn.dept_name||' is : '
 || maxs);
end loop;
END;
/
```

OUTPUT:

```
SQL> CREATE OR REPLACE FUNCTION department_highest (dName Department.dept_name%type)
  2  RETURN Instructor.salary%type as
  3  pop Instructor.salary%type;
  4  BEGIN
  5      select max(salary) into pop
  6      from Instructor group by Instructor.dept_name having Instructor.dept_name in (select dept_name
  7                                                                                    from Instructor
  8                                                                                    where dept_name = dName);
  9      return pop;
 10  END;
 11  /

Function created.

SQL> DECLARE
  2      maxs Instructor.salary%type;
  3      cursor c1 is select distinct dept_name from department;
  4  BEGIN
  5      for dn in c1 loop
  6          maxs := department_highest(dn.dept_name);
  7          dbms_output.put_line('Highest paid salary in '||dn.dept_name||' is : ' || maxs);
  8  end loop;
  9  END;
 10  /
Highest paid salary in Biology is : 72000
Highest paid salary in Comp. Sci. is : 92000
Highest paid salary in Elec. Eng. is : 80000
Highest paid salary in Finance is : 90000
Highest paid salary in History is : 62000
Highest paid salary in Music is : 40000
Highest paid salary in Physics is : 99750

PL/SQL procedure successfully completed.

SQL>
```

Row Triggers

1. **Based on the University database Schema in Lab 2, write a row trigger that records along with the time any change made in the Takes (ID, course-id, sec-id, semester, year, grade) table in log_change_Takes (Time_Of_Change, ID, courseid, sec-id, semester, year, grade).**

CODE:

```
create table log_change_Takes (
        toc timestamp,
        type varchar(3),
```

```
        ID varchar(5),
        course_id varchar(8),
        sec_id varchar(8),
        semester varchar(6),
        year numeric(4,0),
        grade varchar(2),
        primary key (toc, ID, course_id, sec_id, semester, year),
        foreign key (course_id,sec_id, semester, year) references section
        on delete cascade,
        foreign key (ID) references student
        on delete cascade);
CREATE or REPLACE trigger log_change_Takes
BEFORE INSERT OR UPDATE-- OF id,course_id,sec_id,semester,year,grade
OR DELETE on takes
FOR EACH ROW
BEGIN
    CASE
        WHEN INSERTING THEN
        insert into log_change_Takes values (current_timestamp,'ins',:NEW.id,:
NEW.course_id,:NEW.sec_id,:NEW.semester,:NEW.year,:NEW.grade);
        WHEN DELETING THEN
        insert into log_change_Takes values (current_timestamp,'del',:OLD.id,:
OLD.course_id,:OLD.sec_id,:OLD.semester,:OLD.year,:OLD.grade);
        WHEN UPDATING THEN
        insert into log_change_Takes values (current_timestamp,'upd',:NEW.id,:
NEW.course_id,:NEW.sec_id,:NEW.semester,:NEW.year,:NEW.grade);
    END CASE;
END;
/


delete from takes where id = '00128' and course_id = 'CS-101';
insert into takes values ('00128', 'CS-101', '1', 'Fall', '2009', 'A');
update takes set grade = 'B' where id = '98988' and course_id = 'BIO-301';



select * from log_change_Takes;
```

OUTPUT:

```
SQL> create table log_change_Takes (
  2          toc timestamp,
  3          type varchar(3),
  4          ID varchar(5),
  5          course_id varchar(8),
  6          sec_id varchar(8),
  7          semester varchar(6),
  8          year numeric(4,0),
  9          grade varchar(2),
 10          primary key (toc, ID, course_id, sec_id, semester, year),
 11          foreign key (course_id,sec_id, semester, year) references section
 12          on delete cascade,
 13          foreign key (ID) references student
 14          on delete cascade);

Table created.

SQL> CREATE or REPLACE trigger log_change_Takes
  2  BEFORE INSERT OR UPDATE-- OF id,course_id,sec_id,semester,year,grade
  3  OR DELETE on takes
  4  FOR EACH ROW
  5  BEGIN
  6      CASE
  7          WHEN INSERTING THEN
  8          insert into log_change_Takes values (current_timestamp,'ins',:NEW.id,:NEW.course_id,:NEW.sec_id,:NEW.semest
er,:NEW.year,:NEW.grade);
  9          WHEN DELETING THEN
 10          insert into log_change_Takes values (current_timestamp,'del',:OLD.id,:OLD.course_id,:OLD.sec_id,:OLD.semest
er,:OLD.year,:OLD.grade);
 11          WHEN UPDATING THEN
 12          insert into log_change_Takes values (current_timestamp,'upd',:NEW.id,:NEW.course_id,:NEW.sec_id,:NEW.semest
er,:NEW.year,:NEW.grade);
 13      END CASE;
 14  END;
 15  /

Trigger created.

SQL>
```

```
SQL> delete from takes where id = '00128' and course_id = 'CS-101';

1 row deleted.

SQL> insert into takes values ('00128', 'CS-101', '1', 'Fall', '2009', 'A');

1 row created.

SQL> update takes set grade = 'B' where id = '98988' and course_id = 'BIO-301';

1 row updated.

SQL> select * from log_change_Takes;

TOC                                                                         TYP
--------------------------------------------------------------------------- ---
ID    COURSE_I SEC_ID   SEMEST       YEAR GR
----- -------- -------- ------ ---------- --
12-JUN-21 03.11.46.910000 PM                                                del
00128 CS-101   1        Fall         2009 A

12-JUN-21 03.11.46.913000 PM                                                ins
00128 CS-101   1        Fall         2009 A

12-JUN-21 03.11.49.138000 PM                                                upd
98988 BIO-301  1        Summer       2010 B


SQL>
```

2. **Based on the University database schema in Lab: 2, write a row trigger to insert the existing values of the Instructor(ID, name, dept-name, salary) table into a new table Old_ Data_Instructor (ID, name, dept-name, salary) when the salary table is updated.**

CODE:

```
create table old_data_inst(
    id varchar(8),
    name varchar(20),
    dept_name varchar(20),
    salary numeric(8,2),
    primary key (id));
```

```
create or replace trigger old_data_inst
Before update on instructor for each row
begin
    insert into old_data_inst values (:old.id,:old.name,:old.dept_name,:old.sa
lary);
end;
/

update instructor set salary = 91000 where name = 'Wu';

select * from old_data_inst;
```

OUTPUT:

```
SQL> create table old_data_inst(
  2      id varchar(8),
  3      name varchar(20),
  4      dept_name varchar(20),
  5      salary numeric(8,2),
  6      primary key (id));

Table created.

SQL> create or replace trigger old_data_inst
  2  Before update on instructor for each row
  3  begin
  4      insert into old_data_inst values (:old.id,:old.name,:old.dept_name,:old.salary);
  5  end;
  6  /

Trigger created.

SQL>
```

```
SQL> update instructor set salary = 91000 where name = 'Wu';

1 row updated.

SQL> select * from old_data_inst;

ID       NAME                 DEPT_NAME               SALARY
-------- -------------------- -------------------- ----------
12121    Wu                   Finance                 90000

SQL>
```

Database Triggers

3. **Based on the University Schema, write a database trigger on Instructor that checks the following:**
   - **The name of the instructor is a valid name containing only alphabets.**
   - **The salary of an instructor is not zero and is positive.**
   - **The salary does not exceed the budget of the department to which the instructor belongs.**

CODE:

```
CREATE or REPLACE TRIGGER Inst_trig
BEFORE INSERT or UPDATE on Instructor
FOR EACH ROW

DECLARE
bud number(10);

BEGIN

Select budget into bud from department where dept_name=:new.dept_name;
IF :new.name like '%0%' or :new.name like '%1%' or :new.name like '%2%' or :ne
w.name like '%3%' or :new.name like '%4%'
or :new.name like '%5%' or :new.name like '%6%' or :new.name like '%7%' or :ne
w.name like '%8%' or :new.name like '%9%' then
RAISE_APPLICATION_ERROR(-20000,'Insert is denied');
END IF;
IF :new.salary<=0 or :new.salary>bud then
RAISE_APPLICATION_ERROR(-20000,'Insert is denied');
END IF;

END;
/
```

OUTPUT:

```
SQL> CREATE or REPLACE TRIGGER Inst_trig
  2  BEFORE INSERT or UPDATE on Instructor
  3  FOR EACH ROW
  4  DECLARE
  5  bud number(10);
  6  BEGIN
  7  Select budget into bud from department where dept_name=:new.dept_name;
  8  IF :new.name like '%0%' or :new.name like '%1%' or :new.name like '%2%' or :new.name like '%3%' or :new.name like '
%4%'
  9  or :new.name like '%5%' or :new.name like '%6%' or :new.name like '%7%' or :new.name like '%8%' or :new.name like '
%9%' then
 10  RAISE_APPLICATION_ERROR(-20000,'Insert is denied');
 11  END IF;
 12  IF :new.salary<=0 or :new.salary>bud then
 13  RAISE_APPLICATION_ERROR(-20000,'Insert is denied');
 14  END IF;
 15  END;
 16  /

Trigger created.

SQL> insert into instructor values('12345','Testing','History','65945');
insert into instructor values('12345','Testing','History','65945')
            *
ERROR at line 1:
ORA-20000: Insert is denied
ORA-06512: at "SYSTEM.INST_TRIG", line 10
ORA-04088: error during execution of trigger 'SYSTEM.INST_TRIG'
```

4. **Create a transparent audit system for a table Client_master (client_no, name, address, Bal_due). The system must keep track of the records that are being deleted or updated. The functionality being when a record is deleted or modified the original record details and the date of operation are stored in the audit client(client_no, name, bal_due, operation, userid, opdate) table, then the delete or update is allowed to go through.**

CODE:

```
CREATE table client(
    c_no varchar(5) primary key,
    name varchar(20),
    address varchar(100),
    bal_due number);

insert into client values ('01', 'Ayush','Kolkata',10000);
insert into client values ('02', 'Dipesh','Delhi',20000);
insert into client values ('03', 'Rishav','Jaipur',30000);

create table audit_client(
    c_no varchar(5),
    name varchar(20),
    bal_due number,
    op varchar(3),
    user_id varchar(5) default('00000'),
    opDate date);
```

```
create or replace trigger client_audit
BEFORE UPDATE or INSERT on client
```

```
FOR EACH ROW
begin
    case
        WHEN UPDATING THEN
            insert into audit_client values (:OLD.c_no,:OLD.name,:OLD.bal_due,
'upd',NULL,sysdate);
        WHEN DELETING THEN
            insert into audit_client values (:OLD.c_no,:OLD.name,:OLD.bal_due,
'del',NULL,sysdate);
    end case;
end;
/
```

OUTPUT:

```
SQL> CREATE table client(
  2      c_no varchar(5) primary key,
  3      name varchar(20),
  4      address varchar(100),
  5      bal_due number);

Table created.

SQL>
SQL> insert into client values ('01', 'Ayush','Kolkata',10000);

1 row created.

SQL> insert into client values ('02', 'Dipesh','Delhi',20000);

1 row created.

SQL> insert into client values ('03', 'Rishav','Jaipur',30000);

1 row created.

SQL>
SQL> create table audit_client(
  2      c_no varchar(5),
  3      name varchar(20),
  4      bal_due number,
  5      op varchar(3),
  6      user_id varchar(5) default('00000'),
  7      opDate date);

Table created.

SQL>
```

```
SQL> create or replace trigger client_audit
  2  BEFORE UPDATE or INSERT on client
  3  FOR EACH ROW
  4  begin
  5      case
  6          WHEN UPDATING THEN
  7              insert into audit_client values (:OLD.c_no,:OLD.name,:OLD.bal_due,'upd',NULL,sysdate);
  8          WHEN DELETING THEN
  9              insert into audit_client values (:OLD.c_no,:OLD.name,:OLD.bal_due,'del',NULL,sysdate);
 10      end case;
 11  end;
 12  /

Trigger created.

SQL> delete from client where c_no=03;

1 row deleted.
```

```
SQL> select * from Audi_Client;

OPERAT
------
TSTAMP
---------------------------------------------------------------------------
CNO       NAME                 ADDRESS
--------- -------------------- --------------------------------------------------
      DUES USERI
---------- -----
DELETE
12-JUN-21 09.59.41.463000 AM
16796     Loswiueweo           Xvwouire
      6000
```

Instead of Triggers

5. **Based on the University database Schema in Lab 2, create a view Advisor_Student which is a natural join on Advisor, Student and Instructor tables. Create an INSTEAD OF trigger on Advisor_Student to enable the user to delete the corresponding entries in Advisor table.**

CODE:

```
create view Advisor_Student as select s.name s_name, a.S_ID, a.I_ID, i.name i_
name from student s, advisor a, instructor i
where a.S_ID = s.ID and a.I_ID = i.ID;
```

```
create or replace trigger advisor_trigger
instead of delete on Advisor_Student
for each row
begin
delete from advisor where advisor.S_ID = :old.S_ID;
end;
/

delete from Advisor_Student where S_ID = '98988';
```

OUTPUT:

```
SQL> create view Advisor_Student as select s.name s_name, a.S_ID, a.I_ID, i.name i_name from student s, advisor a, instr
uctor i where a.S_ID = s.ID and a.I_ID = i.ID;

View created.

SQL> create or replace trigger advisor_trigger
  2  instead of delete on Advisor_Student
  3  for each row
  4  begin
  5  delete from advisor where advisor.S_ID = :old.S_ID;
  6  end;
  7  /

Trigger created.

SQL> delete from Advisor_Student where S_ID = '98988';

1 row deleted.

SQL> select * from advisor;

S_ID  I_ID
----- -----
00128 45565
12345 10101
23121 76543
44553 22222
45678 22222
76543 45565
76653 98345
98765 98345

8 rows selected.

SQL>
```

THE END