**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:**

**A picture containing shape

Description automatically generated**

**A picture containing text

Description automatically generated**

1. **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 from takes group by course\_id),

studenmod as (select course\_id,student\_count,dept\_name from studentenroll natural join course),

deptmax as (select max(student\_count) as dept\_high,dept\_name from course natural 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:**

**Text

Description automatically generated**

**Functions:**

1. **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:**

**Text

Description automatically generated**

1. **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:**

**Text

Description automatically generated**

**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:**

**Text

Description automatically generated**

**Text

Description automatically generated**

1. **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.salary);

end;

/

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

select \* from old\_data\_inst;

**OUTPUT:**

**Text

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

**Database Triggers**

1. **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 :new.name like '%3%' or :new.name like '%4%'

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

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:**

**Text

Description automatically generated**

1. **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:**

**Text

Description automatically generated**

**Text

Description automatically generated**

**A picture containing text, screenshot

Description automatically generated**

**Instead of Triggers**

1. **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:**

**Text

Description automatically generated**

**THE END**