ST JOSEPH'S UNIVERSITY
BENGALURU . INDIA

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS (MCA)
COURSE CODE: CA 7P2   COURSE TITLE: DBMS LAB (SQL & PL/SQL)

LIST OF PRACTICALS - SQL & PL/SQL

PART A - SQL

1. EXPLORE BUILT-IN FUNCTIONS IN SQL

Exploring the following Built-In Functions in SQL:

(i) NUMBER FUNCTIONS(Group-value functions and List Functions)
(ii) DATE FUNCTIONS
(iii) COUNT FUNCTIONS
(iv) CHARACTER FUNCTION

2. STUDENT ENROLLMENT-BOOK ADOPTION DATABASE MANAGEMENT

Consider the following database of students enrollment in courses and books adopted for each course.

STUDENT(**regno**: string, name: string (not null), major: strong, bdate: date)
COURSE(**course no**: int cname: string (not null), dept: string)
ENROLL(**regno**: string, **course no**: int, sem: int, marks: int)
BOOK-ADOPTION(**course_no**: int, sem: int, **book_isbn**: int)
TEXT(**book isbn**: int, book_title: string, publisher: string, author: string)

(i) Create the above tables by properly specifying the primary keys and the foreign keys
(ii) Enter atleast five tuples for each relation.
(iii) Demonstrate how you can add a new text book to the database and make this book to be adopted by some department.
(iv) Produce a list of text books (include course-no, book_isbn, book-title) in the alphabetical order for courses offered by the **'Compute Science'** department that use more than two books.
(v) List any department that has all its adopted books published by a specific publisher.

3. BOOK DEALER DATABASE MANAGEMENT

The following tables are maintained by a book dealer:

AUTHOR(**author id**: int, name: string, city: string, country: string)
PUBLISHER(**publisher id**: int name: string, city: string, country: string)
CATLOG(**book id**: int, title : string, **author_id**: int, **publisher_id**: int, category: int, year: int, price: int)
CATEGORY(**category id**: int, description: string)
ORDER-DETAILS(**order no**: int, book_id: int, quantity: int)

(i) Create above the tables by properly specifying the primary keys and the foreign keys.

(ii) Enter atleast five tuples for each relation.
(iii) Give the details of the authors who have 2 or more books in the catalog and the price of the books is greater than the average price of the books in the catalog and the year of publication after 2010.
(iv) Find the author of the book that has maximum sales.
(v) Demonstrate how to increase the price of the books published by the specific publisher by 10%

4. BANK DATABASE MANAGEMENT

Consider the following database for BANK.
BRANCH(**ifsc**: string, branch-name: string, branch-city: string, assets: real)
ACCOUNT(**accno**: int, banch-name: string, balance: real)
DEPOSITOR(**accno**: int, customer-name: string)
CUSTOMER(**accno**: int, customer-name: string, customer-street: string, customer-city: string)
LOAN(**loan_no**: int, branch-name: string, amount: real)
BORROWER(**loan_no**: int, customer-name: string)

(i) Create the above tables by properly specifying the primary keys and foreign keys.
(ii) Enter atleast five tuples for each relation.
(iii) Find all the customers who have atleast two accounts at the main branch.
(iv) Find all the customers who have accounts at all the branches located in the specific city.
(v) Demonstrate how to delete all account tuples at every branch located in the specific city.

5. ORDER PROCESSING DATABASE MANAGEMENT

Consider the following database for ORDER PROCESSING.
CUSTOMER(**custno**: int, cname: string, city: string)
ORDER(**orderno**: int, odate: date, ord-amt: real)
ORDER_ITEM(**orderno**: int, itemno:int, qty: int)
ITEM(**itemno**: int, unitprice: real)
SHIPMENT(**orderno**: int, **warehouseno**: int, ship-date: date)
WAREHOUSE(**warehouseno**: int, city: string)

(i) Create the above tables by properly specifying the primary keys and the foreign keys
(ii) Enter atleast five tuples for each relation.
(iii) List the order number and shipping date for all orders that were shipped from the particular warehouse.
(iv) Produce a list of customer details with their name, no. of orders and the average order amount.
(v) List the orders that were not shipped within 30 days from the date of ordering.

6. INSURANCE DATABASE MANAGEMENT

Consider the insurance database given below. The primary keys are underlined and the data types are specified.

PERSON(**driver idno**: string, name: string (not null), address:string)
CAR(**regno**: string, model: string, year: int)
ACCIDENT(**report no**: int, date: date, location: String)
OWNS(**driver idno**: string, **regno**: string)
PARTICIPATED(**driver idno**: string, **regno**: string, report-no: int, damage-amount: int)

(i) Create the above tables by properly specifying the primary keys and the foreign keys
(ii) Enter at least five tuples for each relation.
(iii)Demonstrate how you can;
   a. update the cost of damage for the car that met with accident in report no. 12 as Rs 25000 along with the register number.
   b. Add a new accident happened into the database.
(iv) Find the total number of people who owned cars that were involved in accidents in 2022
(v) Find the number of accidents in which cars of a specific model were involved.

PART B - PL/SQL

7. MENU DRIVEN PL/SQL PROGRAM FOR COMPUTING STUDENT'S AVERAGE, RESULT AND GRADE AND ALSO FINDING FIRST, SECOND AND THIRD AVERAGE STUDENT'S DETAILS

Create the following tables and insert atleast five tuple in Student Table:
Student (**Rno**: string, Name: string (not null), Sub1: number, Sub2: number, Sub3 : number, Sub4 : number, Sub5 : number, Average : number, Result : string, Grade : string)

StudentPass (**Rno**: string, Name: string (not null), Sub1: number, Sub2: number, Sub3 : number, Sub4 : number, Sub5 : number, Average : number, Result : string, Grade : string)

StudentFail (**Rno**: string, Name: string (not null), Sub1: number, Sub2: number, Sub3 : number, Sub4 : number, Sub5 : number, Average : number, Result : string, Grade : string)

Write a **menu driven PL/SQL program** to (i) compute average and grade of students and split the table into two such as StudentPass and StudentFail **(Specify the passed and failed students separately)** and
(ii) display first, second and third average details of students

SQL COMMANDS

```
create table student2(Rno varchar2(10),Name varchar2(35),
Sub1 number(3),Sub2 number(3),Sub3 number(3),Sub4 number(3),
Sub5 number(3),Average number(6,2),Result varchar2(10),
Grade varchar2(30),constraint pk_student primary key(Rno));
```

```
insert into student2(Rno,Name,Sub1,Sub2,Sub3,Sub4,Sub5)
values('&Rno','&Name',&Sub1,&Sub2,&Sub3,&Sub4,&Sub5);

alter table student2 modify Name varchar2(35);

alter table student2 modify Grade varchar2(30);
```

PL/SQL PROGRAM

```
DECLARE
   CURSOR c  IS  SELECT  *  FROM  student2;
   x student2%rowtype;
   res student2.result%type;
   per student2.average%type;
   gra student2.grade%type;
   ch number;
BEGIN
   DELETE FROM student2pass;
   DELETE FROM student2fail;
   DBMS_OUTPUT.PUT_LINE('1.COMPUTE AVERAGE AND GRADE OF
STUDENTS');
   DBMS_OUTPUT.PUT_LINE('2.DISPLAY FIRST, SECOND AND THIRD
AVERAGE DETAILS OF STUDENTS');
   DBMS_OUTPUT.PUT_LINE('3.EXIT');
   ch:=&ch;
   DBMS_OUTPUT.PUT_LINE('Enter your choice : '||ch);

    IF (ch=1) THEN
        FOR x IN c
        LOOP
            IF (x.sub1>=40 and x.sub2>=40 and x.sub3>=40
                and x.sub4>=40 and x.sub5>=40) THEN
                res:='PASS';
                per:=(x.sub1+x.sub2+x.sub3+x.sub4+x.sub5)/5;
                DBMS_OUTPUT.PUT_LINE('The percentage is ' || per);
                IF per>=75 THEN
                   gra:='FIRST CLASS WITH DISTINCTION';
                ELSIF per<75 and per>=60 THEN
                   gra:='FIRST CLASS';
                ELSIF per<60 and per>=50 THEN
                   gra:='SECOND CLASS';
                ELSE
                   gra:='THIRD CLASS';
                END IF;
                UPDATE student2 SET result=res WHERE rno=x.rno;
                UPDATE student2 SET average=per WHERE rno=x.rno;
            UPDATE student2 SET grade=gra WHERE rno=x.rno;
            INSERT INTO student2pass VALUES(x.rno,x.name,
               x.sub1,x.sub2,x.sub3,x.sub4,x.sub5,per,res,gra);
            DBMS_OUTPUT.PUT_LINE('RECORD        UPDATED        AND
SPLITTED');
         ELSE
            res:='FAIL';
            UPDATE student2 SET result=res WHERE rno=x.rno;
            INSERT INTO student2fail VALUES(x.rno,x.name,
               x.sub1,x.sub2,x.sub3,x.sub4,x.sub5,per,res,gra);
            DBMS_OUTPUT.PUT_LINE('RECORD        UPDATED        AND
SPLITTED');
         END IF;
      END LOOP;
   END IF;
   IF (CH=2) THEN
      SELECT * INTO x FROM student2 WHERE average=
         (SELECT MAX(average) FROM student2);
      DBMS_OUTPUT.PUT_LINE ('FIRST AVERAGE STUDENT DETAILS');
      DBMS_OUTPUT.PUT_LINE ('============================');
      DBMS_OUTPUT.PUT_LINE ('REGNO: '|| x.rno);
      DBMS_OUTPUT.PUT_LINE ('NAME: ' || x.name);
      DBMS_OUTPUT.PUT_LINE ('AVERAGE: ' || x.average);

      SELECT * INTO x FROM student2 WHERE average=
         (SELECT MAX(average) FROM student2
         WHERE average < (SELECT MAX(average) FROM student2));
      DBMS_OUTPUT.PUT_LINE     ('SECOND     AVERAGE     STUDENT
DETAILS');
      DBMS_OUTPUT.PUT_LINE ('============================');
      DBMS_OUTPUT.PUT_LINE ('REGNO: '|| x.rno);
      DBMS_OUTPUT.PUT_LINE ('NAME: ' || x.name);
      DBMS_OUTPUT.PUT_LINE ('AVERAGE: ' || x.average);

      SELECT * INTO x FROM student2 WHERE average=
         (SELECT MAX(average) FROM student2
         WHERE average < (SELECT MAX(average) FROM student2
         WHERE average < (SELECT MAX(average) FROM student2)));
      DBMS_OUTPUT.PUT_LINE ('THIRD AVERAGE STUDENT DETAILS');
      DBMS_OUTPUT.PUT_LINE ('============================');
      DBMS_OUTPUT.PUT_LINE ('REGNO: '|| x.rno);
      DBMS_OUTPUT.PUT_LINE ('NAME: ' || x.name);
      DBMS_OUTPUT.PUT_LINE ('AVERAGE: ' || x.average);
   END IF;
EXCEPTION
  WHEN no_data_found THEN
```

```
    DBMS_OUTPUT.PUT_LINE('No  Records!');
  WHEN others THEN
    DBMS_OUTPUT.PUT_LINE('Error!');
END;
/
```

8. MENU DRIVEN PL/SQL PROGRAM TO COMPUTE FACTORIAL VALUE AND GENERATE FIBONACCI SERIES USING RECURSIVE FUNCTIONS

Write a **menu driven PL/SQL program** using recursive functions to (i) compute factorial value of a given number and (ii) generate Fibonacci series of the given number of terms

PL/SQL RECURSIVE FUNCTIONS

```
create or replace function fact(n number)
return number is
begin
   if (n=0) then
     return(1);
   else
     return(n*fact(n-1));
   end if;
end;
/

create or replace function fibo(n number)
return number is
begin
   if (n=1) then
     return(0);
   elsif (n=2) then
     return(1);
   else
     return(fibo(n-1)+fibo(n-2));
   end if;
end;
/
```

PL/SQL MAIN PROGRAM

```
declare
  n number;
  ch number;
begin
  dbms_output.put_line('1.FACTORIAL VALUE');
  dbms_output.put_line('2.FIBONACCI SERIES');
  dbms_output.put_line('3.EXIT');
  ch:=&ch;
  dbms_output.put_line('Enter your choice : '||ch);
  n:=&n;
  case ch
    when 1 then
      dbms_output.put_line('Factorial Value of '||n||' is '||fact(n));
    when 2 then
    begin
      dbms_output.put_line('Fibonacci Series are');
      for i in 1..n
      loop
        dbms_output.put_line(fibo(i));
      end loop;
    end;
    else
      dbms_output.put_line('THANK YOU');
   end case;
end;
/
```

9. MENU DRIVEN PL/SQL PROGRAM USING PROCEDURES TO GET EMPLOYEE DETAILS BY EID, FIND FIRST, SECOND, THIRD MAX SALARY DETAILS OF EMPLOYEE AND FIND FIRST SENIOR MOST, SECOND SENIOR MOST EMPLOEE DETAILS

Create the following table and insert atleast five tuples in it:
Emp (**eid** : String, ename : String, dob: date, doj: date, salary: number (salary>0))

Write a **menu driven PL/SQL program** using procedures to (i) get employee details by eid, (ii) get employee details of first highest salary,
(iii) get employee details of second highest salary, (iv) get employee details of third highest salary, (v) get first senior most employee details based on date of birth (dob) of the employee and (vi) get second senior most employee details based on date of joining (doj) of the employee.

SQL COMMANDS

```
create table emp(eid varchar2(12) primary key, ename varchar2(50),
dob date, salary number(16,2) check (salary>0));

insert into emp values('&eid','&ename','&dob',&salary);
```

PL/SQL PROCEDURES

```
create or replace procedure getByEID(id in emp.eid%type,
  ename out emp.ename%type, salary out emp.salary%type)
is
begin
     select ename, salary into ename, salary from emp where eid=id;
end;
/

create or replace procedure findFirstMaxSalary(eid out emp.eid%type,
  ename out emp.ename%type, dob out emp.dob%type,
  salary out emp.salary%type)
is
begin
     select eid, ename, dob, salary into eid, ename, dob, salary
     from emp where salary=(select max(salary) from emp);
end;
/

create or replace procedure findSecondMaxSalary(eid out emp.eid%type,
  ename out emp.ename%type, dob out emp.dob%type,
  salary out emp.salary%type)
is
begin
     select eid, ename, dob, salary into eid, ename, dob, salary
     from emp where salary=(select max(salary) from emp
     where salary < (select max(salary) from emp));
end;
/

create or replace procedure findThirdMaxSalary(eid out emp.eid%type,
  ename out emp.ename%type, dob out emp.dob%type,
  salary out emp.salary%type)
is
begin
     select eid, ename, dob, salary into eid, ename, dob, salary
     from emp where salary=(select max(salary) from emp
     where salary < (select max(salary) from emp
     where salary < (select max(salary) from emp)));
end;
/

create or replace procedure findFirstSeniorMost(eid out emp.eid%type,
  ename out emp.ename%type, dob out emp.dob%type,
  salary out emp.salary%type)
is
begin
     select eid, ename, dob, salary into eid, ename, dob, salary
     from emp where dob=(select min(dob) from emp);
end;
/

create or replace procedure findSecondSeniorMost(eid out emp.eid%type,
  ename out emp.ename%type, dob out emp.dob%type,
  salary out emp.salary%type)
is
begin
     select eid, ename, dob, salary into eid, ename, dob, salary
     from emp where dob=(select min(dob) from emp
     where dob > (select min(dob) from emp));
end;
/
```

PL/SQL MAIN PROGRAM

```
declare
    eid emp.eid%type;
    ename emp.ename%type;
    dob emp.dob%type;
    salary emp.salary%type;
    ch number;
begin
    dbms_output.put_line('1.GET EMPLOYEE DETAILS BY EID');
    dbms_output.put_line('2.GET EMPLOYEE DETAILS OF FIRST
HIGHEST SALARY');
    dbms_output.put_line('3.GET EMPLOYEE DETAILS OF SECOND
HIGHEST SALARY');
    dbms_output.put_line('4.GET EMPLOYEE DETAILS OF THIRD
HIGHEST SALARY');
    dbms_output.put_line('5.GET FIRST SENIOR MOST EMPLOYEE
DETAILS');
    dbms_output.put_line('6.GET SECOND SENIOR MOST EMPLOYEE
DETAILS');
    dbms_output.put_line('7.EXIT');
    ch:=&ch;
    dbms_output.put_line('Enter your choice : '||ch);
    if (ch=1) then
        eid:='&eid';
        dbms_output.put_line('EMPLOYEE DETAILS BY EID');
        dbms_output.put_line('**********************');
        getByEID(eid,ename,salary);
        dbms_output.put_line('EID: '||eid);
        dbms_output.put_line('ENAME:  '||ename);
        dbms_output.put_line('SALARY:  '||salary);
```

```
  elsif (ch=2) then
    dbms_output.put_line('EMPLOYEE   DETAILS   OF   FIRST   HIGEST
SALARY');
    dbms_output.put_line('***************************************');
    findFirstMaxSalary(eid,ename,dob,salary);
    dbms_output.put_line('EID: '||eid);
    dbms_output.put_line('ENAME:  '||ename);
    dbms_output.put_line('DATE OF BIRTH:  '||dob);
    dbms_output.put_line('SALARY:  '||salary);
  elsif (ch=3) then
    dbms_output.put_line('EMPLOYEE DETAILS OF SECOND HIGEST
SALARY');
    dbms_output.put_line('****************************************');
    findSecondMaxSalary(eid,ename,dob,salary);
    dbms_output.put_line('EID: '||eid);
    dbms_output.put_line('ENAME:  '||ename);
    dbms_output.put_line('DATE OF BIRTH:  '||dob);
    dbms_output.put_line('SALARY:  '||salary);
  elsif (ch=4) then
    dbms_output.put_line('EMPLOYEE DETAILS OF THIRD HIGEST
SALARY');
    dbms_output.put_line('***************************************');
    findThirdMaxSalary(eid,ename,dob,salary);
    dbms_output.put_line('EID: '||eid);
    dbms_output.put_line('ENAME:  '||ename);
    dbms_output.put_line('DATE OF BIRTH:  '||dob);
    dbms_output.put_line('SALARY:  '||salary);
  elsif (ch=5) then
    dbms_output.put_line('FIRST     SENIOR     MOST     EMPLOYEE
DETAILS');
    dbms_output.put_line('**********************************');
    findFirstSeniorMost(eid,ename,dob,salary);
    dbms_output.put_line('EID: '||eid);
    dbms_output.put_line('ENAME:  '||ename);
    dbms_output.put_line('DATE OF BIRTH:  '||dob);
    dbms_output.put_line('SALARY:  '||salary);
  elsif (ch=6) then
    dbms_output.put_line('SECOND     SENIOR     MOST     EMPLOYEE
DETAILS');
    dbms_output.put_line('**********************************');
    findSecondSeniorMost(eid,ename,dob,salary);
    dbms_output.put_line('EID: '||eid);
    dbms_output.put_line('ENAME:  '||ename);
    dbms_output.put_line('DATE OF BIRTH:  '||dob);
    dbms_output.put_line('SALARY:  '||salary);
  else
```

```
        dbms_output.put_line('THANK YOU! GOOD BYE!!');
     end if;
exception
   when no_data_found then
      dbms_output.put_line('No Records!');
   when others then
      dbms_output.put_line('Error!');
end;
/
```

10. MENU DRIVEN PL/SQL PROGRAM USING PROCEDURES AND FUNCTIONS OF CUSTOM PACKAGE TO FIND AREA AND CIRCUMFERENCE OF A CIRCLE AND GCD OF TWO NUMBERS

Write a **menu driven PL/SQL program** using procedures and functions of **custom package** to (i) find area and circumference of a circle **(using procedure)** and (ii) GCD (Greatest Common Divisor) of the given two numbers **(using recursive function)**

PL/SQL CUSTOM PACKAGE AND ITS BODY CREATION

```
--Creating a package called mypkg
create or replace package mypkg
as
      --Procedure  Declaration
      procedure area_circum_of_circle (r in number, a out number, c out
number);
      --Function Declaration
      function gcd(a number, b number) return number;
end mypkg;

--Creating a package body called mypkg
create or replace package body mypkg
as
       --Procedure definition
       procedure area_circum_of_circle (r in number, a out number, c out
number)
       is
       begin
           declare
              pi constant number(10,4):=3.1416;
           begin
              a:=pi * (r * r);
              c:=2 * pi * r;
           end;
       end area_circum_of_circle;
       --Function definition
```

```
        function gcd(a number, b number)
        return number is
        begin
             if b = 0 then
               return a;
           else
               return gcd(b, a mod b);
           end if;
        end gcd;
end mypkg;
```

PL/SQL MAIN PROGRAM

```
--Invoking Functions/Procedures of a Package called mypkg
declare
   r  number;
   a  number;
   c  number;
   x  number;
   y number;
   ch number;
begin
   dbms_output.put_line('1.CALLING THE PROCEDURE
AREA_CIRCUM_OF_CIRCLE IN THE PACKAGE');
   dbms_output.put_line('2.CALLING THE FUNCTION GCD IN THE
PACKAGE');
   dbms_output.put_line('3.EXIT');
   ch:=&ch;
   dbms_output.put_line('Enter your choice : '||ch);
   if (ch=1) then
       r:=&r;
       dbms_output.put_line('Enter the radius of a circle: '||r);
       mypkg.area_circum_of_circle(r,a,c);
       dbms_output.put_line('Area of Circle is '||a);
       dbms_output.put_line('Circumference of Circle is '||c);
   elsif (ch=2) then
       x:=&x;
       dbms_output.put_line('Enter the value for x: '||x);
       y:=&y;
       dbms_output.put_line('Enter the value for y: '||y);
       dbms_output.put_line('Greatest Common Divisor (GCD) of '||x||'
and '||y||' is '||mypkg.gcd(x,y));
   else
       dbms_output.put_line('THANK YOU');
   end if;
end;
```

11. EMPLOYEE DATABASE MANAGEMENT USING TRIGGERS

Create the following tables:

employee (**<u>eid</u>**: string, ename : string (not null), salary : number (salary>0));
employee10000 (**<u>eid</u>**: string, ename : String (not null), salary : number (salary>0), msg : string);

Create two triggers: One for automatically inserting employee records into **employee10000** table when we insert or update employee records in **employee** table **(if an employee's salary is greater than 10000)** and the **other one** for automatically inserting employee records into **employee10000** table when we delete employee records from **employee** table.

<u>SQL COMMANDS AND PL/SQL TRIGGERS</u>

<u>SQL COMMANDS FOR TABLES CREATION</u>

```
create table employee(eid varchar2(6) primary key,
ename varchar2(35) not null, salary number(15,2)
check (salary>0))
/

create table employee10000(eid varchar2(6) primary key,
ename varchar2(35) not null, salary number(15,2)
check (salary>0), msg varchar2(60))
/
```

<u>PL/SQL TRIGGERS</u>

```
create or replace trigger employee_trig
before insert or update on  employee
for each row
when (new.salary>10000)
begin
    if inserting then
            insert into employee10000 values
(:new.eid,:new.ename,:new.salary,'RECORD INSERTED INTO employee');
            dbms_output.put_line('RECORD INSERTED INTO
EMPLOYEE10000 TABLE');
    end if;
    if updating then
            insert into employee10000 values
(:new.eid,:new.ename,:new.salary,'RECORD UPDATED ON employee');
```

```
        dbms_output.put_line('RECORD        INSERTED        INTO
EMPLOYEE10000 TABLE');
   end if;
end employee_trig;
/

create or replace trigger employee_trig_del
after delete on employee
for each row
when (old.salary>10000)
begin
    if deleting then
        --delete from employee10000 where eid=:old.eid;
        insert        into        employee10000        values
(:old.eid,:old.ename,:old.salary,'RECORD DELETED FROM employee');
        dbms_output.put_line('RECORD        DELETED        FROM
EMPLOYEE10000 TABLE');
   end if;
end employee_trig_del;
/
```

SQL COMMANDS FOR TESTING THE TRIGGERS

```
SQL> insert into employee values('&eid','&ename',&salary);
Enter value for eid: E101
Enter value for ename: RAMA L
Enter value for salary: 67898.98
old   1: insert into employee  values('&eid','&ename',&salary)
new   1: insert into employee values('E101','RAMA L',67898.98)
RECORD INSERTED INTO EMPLOYEE10000 TABLE

1 row created.

SQL> /
Enter value for eid: E102
Enter value for ename: BALU D
Enter value for salary: 3545.56
old   1: insert into employee values('&eid','&ename',&salary)
new   1: insert into employee values('E102','BALU D',3545.56)

1 row created.

SQL> /
Enter value for eid: E103
Enter value for ename: THANGAM S
Enter value for salary: 12456.78
```

```
old   1: insert into employee values('&eid','&ename',&salary)
new   1: insert into employee values('E103','THANGAM S',12456.78)
RECORD INSERTED INTO EMPLOYEE10000 TABLE

1 row created.

SQL> select * from employee;

EID    ENAME                                  SALARY
------------------------------------------------------- ----------------------
E101   RAMA L                               67898.98
E102   BALU D                                3545.56
E103   THANGAM S                               12456.78

SQL> select * from employee10000;

EID    ENAME                                  SALARY
------------------------------------------------------- ----------------------
E101   RAMA L                               67898.98
E103   THANGAM S                               12456.78

SQL> update employee set salary=12300.78 where eid='E102';
RECORD INSERTED INTO EMPLOYEE10000 TABLE

1 row updated.

SQL> select * from employee;

EID    ENAME                                  SALARY
------------------------------------------------------- ----------------------
E101   RAMA L                               67898.98
E102   BALU D                               12300.78
E103   THANGAM S                               12456.78

SQL> select * from employee10000;

EID    ENAME                                  SALARY
------------------------------------------------------- ----------------------
E101   RAMA L                               67898.98
E103   THANGAM S                               12456.78
E102   BALU D                               12300.78

SQL> delete from employee where eid='E101';
RECORD DELETED FROM EMPLOYEE10000 TABLE

1 row deleted.
```

```
SQL> select * from employee;

EID    ENAME                              SALARY
---------------------------------------------------------- ------------------------
E102   BALU D                            12300.78
E103   THANGAM S                           12456.78

SQL> select * from employee10000;

EID    ENAME                              SALARY
---------- ------------------------------------------------------------ -----------------
E103   THANGAM S                          12456.78
E102   BALU D                            12300.78

SQL> delete from employee where eid='E102';
RECORD DELETED FROM EMPLOYEE10000 TABLE

1 row deleted.

SQL> select * from employee;

EID    ENAME                              SALARY
---------- ------------------------------------------------------------ -----------------
E103   THANGAM S                          12456.78

SQL> select * from employee10000;

EID    ENAME                              SALARY
---------- ------------------------------------------------------------ -----------------
E103   THANGAM S                          12456.78

SQL>
```

12. STUDENT DATABASE MANAGEMENT USING TRIGGERS

Create the following tables:

Student (**Rno**: String, Name: String (not null), Sub1: number, Sub2: number, Sub3 : number, Sub4 : number, Sub5 : number, Average : number, Result : String, Grade : String)

StudentPass (**Rno**: String, Name: String (not null), Sub1: number, Sub2: number, Sub3 : number, Sub4 : number, Sub5 : number, Average : number, Result : String, Grade : String)

StudentFail (**<u>Rno</u>**: String, Name: String (not null), Sub1: number, Sub2: number, Sub3 : number, Sub4 : number, Sub5 : number, Average : number, Result : String, Grade : String)

Create two triggers: One for automatically calculating and updating the average and result of the student and inserting the student record into **StudentPass** table if his/her result is **“pass”** and insert the student record into **StudentFail** table if his/her result is **“fail”** when we insert or update student records of student table and the other one for automatically deleting the student record from either **StudentPass** table or **StudentFail** table when we delete any student's record from **Student** table.

SQL COMMANDS AND PL/SQL TRIGGERS

SQL COMMANDS FOR TABLES CREATION

```
create table student2(Rno varchar2(10),Name varchar2(35),
Sub1 number(3),Sub2 number(3),Sub3 number(3),Sub4 number(3),
Sub5 number(3),Average number(6,2),Result varchar2(10),
Grade varchar2(30),constraint pk_student primary key(Rno));

create table student2pass(Rno varchar2(10), Name varchar2(35),
Sub1 number(3),Sub2 number(3),Sub3 number(3),Sub4 number(3),
Sub5 number(3),Average number(6,2),Result varchar2(10),
Grade varchar2(30),constraint pk_student_pass primary key(Rno));

create table student2fail(Rno varchar2(10), Name varchar2(35),
Sub1 number(3),Sub2 number(3),Sub3 number(3),Sub4 number(3),
Sub5 number(3),Average number(6,2),Result varchar2(10),
Grade varchar2(30),constraint pk_student_fail primary key(Rno));
```

PL/SQL TRIGGERS

```
create or replace trigger student_trig
before insert or update on student2
for each row
begin
  declare
     res student2.result%type;
     per student2.average%type;
     gra student2.grade%type;
  begin
     delete from student2pass where rno=:new.rno;
     delete from student2fail where rno=:new.rno;
     if inserting or updating then
```

```
        if (:new.sub1>=40 and :new.sub2>=40 and :new.sub3>=40
           and :new.sub4>=40 and :new.sub5>=40) then
           res:='PASS';

per:=(:new.sub1+:new.sub2+:new.sub3+:new.sub4+:new.sub5)/5;
            dbms_output.put_line('The percentage is ' || per);
            if per>=75 then
                  gra:='FIRST CLASS WITH DISTINCTION';
            elsif per<75 and per>=60 then
                gra:='FIRST CLASS';
            elsif per<60 and per>=50 then
                gra:='SECOND CLASS';
            else
                gra:='THIRD CLASS';
            end if;
            update student2 set result=res where rno=:new.rno;
            update student2 set average=per where rno=:new.rno;
            update student2 set grade=gra where rno=:new.rno;
            insert into student2pass values(:new.rno,:new.name,

:new.sub1,:new.sub2,:new.sub3,:new.sub4,:new.sub5,per,res,gra);
            dbms_output.put_line('RECORD UPDATED AND SPLITTED');
         else
            res:='FAIL';
            update student2 set result=res where rno=:new.rno;
            insert into student2fail values(:new.rno,:new.name,

:new.sub1,:new.sub2,:new.sub3,:new.sub4,:new.sub5,per,res,gra);
            dbms_output.put_line('RECORD UPDATED AND SPLITTED');
         end if;
     end if;
  exception
    when no_data_found then
       dbms_output.put_line('No records found!');
  end;
end student_trig;
/

create or replace trigger student_trig_del
after delete on student2
for each row
begin
     if deleting then
         delete from student2pass where rno=:old.rno;
         delete from student2fail where rno=:old.rno;
        dbms_output.put_line('RECORD DELETED FROM STUDENT2PASS
AND STUDENT2FAIL TABLES');
    end if;
end student_trig_del;
/
```

SQL COMMANDS FOR TESTING THE TRIGGERS

```
insert into student2(Rno,Name,Sub1,Sub2,Sub3,Sub4,Sub5)
values('&Rno','&Name',&Sub1,&Sub2,&Sub3,&Sub4,&Sub5);

set linesize 180;

select * from student2;

select * from student2pass;

select * from student2fail;
```

13. SALES COMMISSION COMPUTATION USING TRIGGERS

Create the following tables:

salesmen (**smno** : string, name : string (not null), act_sales_amt : number, tgt_sales_amt : number);

salescommission(**smno** : string, name (not null) : string, dos: date, comm : number);

Create two triggers: One for automatically calculating the salescommision if **act_sales_amt>tgt_sales_amt** then the trigger is calculating the sales commission for the salesman based on the following criteria:

```
if act_sales_amt>25000 then
   comm:= act_sales_amt * 0.35;
else if act_sales_amt>20000 and act_sales_amt<=25000 then
   comm:= act_sales_amt * 0.25;
else if act_sales_amt>15000 and act_sales_amt<=20000 then
   comm:= act_sales_amt * 0.15;
else if act_sales_amt>10000 and act_sales_amt<=15000 then
   comm:= act_sales_amt * 0.10;
else raise less_than_fixed exception;
```

and insert that **salescommission** record into **salescommission** table when we insert or update salesman records of **salesmen** table and the other one for automatically deleting salesmen's commission record from

salescommission table when we delete salesman records from **salesman** table.

SQL COMMANDS AND PL/SQL TRIGGERS

SQL COMMANDS FOR TABLES CREATION

```
create table salesmen(smno varchar2(6) primary key, name varchar2(50), act_sales_amt number(16,2), tgt_sales_amt number(16,2));

create table salescommission(smno varchar2(6), name varchar2(50), dos date, comm number(14,2));

drop trigger  salesman_trig;

drop  trigger  salesman_trig_del;
```

PL/SQL TRIGGERS

```
create or replace trigger salesmen_trig
before insert or update  on  salesmen
for each row
when (new.act_sales_amt>new.tgt_sales_amt)
begin
  declare
     comm salescommission.comm%type;
     less_than_fixed exception;
     na salesmen.name%type;
  begin
     if :new.act_sales_amt>25000 then
        comm:= :new.act_sales_amt * 0.35;
     elsif :new.act_sales_amt>20000 and :new.act_sales_amt<=25000 then
        comm:= :new.act_sales_amt * 0.25;
     elsif :new.act_sales_amt>15000 and :new.act_sales_amt<=20000 then
        comm:= :new.act_sales_amt * 0.15;
     elsif :new.act_sales_amt>10000 and :new.act_sales_amt<=15000 then
        comm:= :new.act_sales_amt * 0.10;
     else
        raise less_than_fixed;
     end if;
     if inserting then
         insert into salescommission values (:new.smno,:new.name,sysdate,comm);
         dbms_output.put_line('RECORD INSERTED INTO SALESCOMMISSION TABLE');
     end if;
     if updating then
           select name into na from salescommission where
smno=:new.smno;
           delete from salescommission where smno=:new.smno;
           insert into salescommission values
(:new.smno,:new.name,sysdate,comm);
           dbms_output.put_line('RECORD INSERTED INTO
SALESCOMMISSION TABLE');
     end if;
  exception
    when less_than_fixed then
      dbms_output.put_line('salesman no.: '||:new.smno||' is not entitled
to get commission');
    when no_data_found then
      dbms_output.put_line('The Salesman '||na||' is not found in the
salescommission table');
  end;
end salesmen_trig;
/

create or replace trigger salesmen_trig_del
after delete on salesmen
for each row
begin
  declare
     na salesmen.name%type;
  begin
     if deleting then
           select name into na from salescommission where
smno=:old.smno;
           delete from salescommission where smno=:old.smno;
           dbms_output.put_line('RECORD DELETED FROM
SALESCOMMISSION TABLE');
     end if;
  exception
     when no_data_found then
      dbms_output.put_line('salesman no.: '||:old.smno||' is not found in
the salescommission table');
  end;
end salesmen_trig_del;
/
```

SQL COMMANDS FOR TESTING THE TRIGGERS

```
insert into salesmen values('&smno','&name',&act_sales_amt,
&tgt_sales_amt);
```

```
select * from salesmen;

select * from salescommission;

update salesmen set act_sales_amt=18900, tgt_sales_amt=10000 where smno='SM01';

update salesmen set act_sales_amt=18900 where smno='SM01';

delete from salesmen where smno='SM02';
```

INSTRUCTIONS TO THE STUDENTS	
1	Students should write the **TITLE OF THE PROGRAM, AIM, CODING (SQL COMMANDS or PL/SQL PROGRAMS / PROCEDURES / FUNCTIONS / PACKAGES (Write whichever is applicable to the exercise)), INPUT / OUTPUT and RESULT** for every lab exercise **(They should write at least 3 lab exercises per lab session)** in a **long size OBSERVATION NOTE BOOK (ONB)** before **entering to the lab. Please maintain separate ONB for every lab course.**
2	After the successful completion of the lab exercises and getting signatures from the concerned teacher in the ONB, the students should also write the **TITLE OF THE PROGRAM, AIM, CODING (SQL COMMANDS or PL/SQL PROGRAMS / PROCEDURES / FUNCTIONS / PACKAGES (Write whichever is applicable to the exercise)), INPUT / OUTPUT and RESULT** for every lab exercise in a **RECORD NOTE BOOK (RNB). Please maintain separate RECORD NOTE BOOK for every lab course.**