-- PL/SQL Practical --

-- Creating tables --

-- The following schema is required by a computer training institute to store information:

-- 1> Course table: holds information about available courses.

```
create table course (
coursecode  number(2)not null,
coursename varchar2(30),
syllabus varchar2(60)
);
```

-- 2> Batch table: holds info about batches. 1 course can have multiple batches.

```
create table batch (
batchcode number(5) not null,
coursecode number(2) not null,
startingdate date not null,
duration number(3) not null,
coursefees number(10,2) not null,
netincome number(10,2),
expectedincome number(10,2)
);
```

--3> Advertisement table: the institute lists advertisements in newspaper.

```
create table advertisement (
refcode number(5) not null
,medianame varchar2(30)
,pages number(2)
,advdate date not null
,amount number(10,2) );
```

--4> Enquiry table: in response to advertisement, institute gets response in form of enquiries.

```
create table enquiry (
enquiryno number(8) not null
,fname varchar2(30)
,sname varchar2(30)
,coursecode number(2)
,plotno varchar2(30)
,street varchar2(30)
,city varchar2(30)
,pincode number(6)
,phone number(6)
,enquirydate date not null
,refcode number(5)
,en_st char(1));
```

--5> Enrollment table: Information about enrollments. Only some enquiries are converted into enrollments.

```
create table enrollment (
rollno number(8) not null
,enquiryno number(8)
,batchcode number(5)
,enrollmentdate date not null );
```

--6> Installment table: Students can pay fees in installments.

```
create table installment (
rollno number(8)
,installmentdate date not null
,amount number(10,2) not null
,installmentstatus char(1) );
```

-- 7> feespaid table:

```
create table feespaid (
rollno number(8) not null,
feespaiddate date not null,
chequeno varchar2(10),
bankname varchar2(30),
remarks varchar2(30),
amount number(10,2));
```

-- 8> Expenditure table: holds expenditure details of the institute.

```
create table EXPENDITURE (
expcode number(5)
,amount number(10,2)
,expdate date
,chequeno varchar2(30)
,bankname varchar2(30)
,remark varchar2(100)
,batchcode number(5));
```

--9> Master Expenditure table: holds total of all types of expenditure.

```
create table expmaster (
expcode number(5) not null
,expname varchar2(30)
,exptype varchar2(10)
,netexp number(10,2));
----------------------------------
```

```
-- describe all tables.
desc course;
desc batch;
desc advertisement;
desc enquiry;
desc enrollment;
desc installment;
desc feespaid;
desc expenditure;
desc expmaster;
-----------------------------------

-- insert records into all the tables:
INSERT INTO COURSE VALUES (10,'ORACLE','SQL,PLSQL,JDBC,JSP');
INSERT INTO COURSE VALUES (20,'JAVA','JAVA_Core, JAVA_Advanced,SERVLETS,JSP');
INSERT INTO BATCH VALUES (20002,20,'01-JUL-16',180,24000,0,0);
INSERT INTO BATCH VALUES (10002,10,'23-JAN-16',3,6000,2000,0);
INSERT INTO BATCH VALUES (10001,10,'11-JAN-16',2,7000,4600,6000);
INSERT INTO BATCH VALUES (10003,10,'25-JAN-16',4,6000,8765,7000);

INSERT INTO ADVERTISEMENT (REFCODE,MEDIANAME,ADVDATE)
VALUES (1001,'Times Of India','12-MAR-16');

INSERT INTO ADVERTISEMENT
(REFCODE,MEDIANAME,ADVDATE)
VALUES (1002,'Hindustan Times','15-APR-16');
```

```
INSERT INTO ENQUIRY (ENQUIRYNO,FNAME,SNAME,COURSECODE,
PLOTNO,STREET,CITY,ENQUIRYDATE,REFCODE,EN_ST) VALUES
(110001,'ANIL','SHARMA',10,'23D','RAMNAGAR','NAGPUR','01-JUL-15',1001,'Y');

INSERT INTO ENQUIRY (ENQUIRYNO,FNAME,SNAME,COURSECODE,
PLOTNO,STREET,CITY,ENQUIRYDATE,REFCODE,EN_ST) VALUES (110002,'SACHIN','NIMJE',20,
'20','GANDHI NAGAR','NAGPUR','01-SEP-16',1002,'Y');


INSERT INTO ENROLLMENT VALUES (20002001,100826,20002,'01-JUL-16');

INSERT INTO ENROLLMENT VALUES (20002003, 100827, 20002, '11-JUL-16');

INSERT INTO ENROLLMENT VALUES (20002009, 100844, 20002, '01-JUL-16');

INSERT INTO ENROLLMENT VALUES (20002005, 100830, 20002, '01-JUL-16');

INSERT INTO ENROLLMENT VALUES (20002010, 100845, 20002, '01-JUL-16' );

INSERT INTO ENROLLMENT VALUES (10001100 ,110001,10001,'01-DEC-16');


INSERT INTO INSTALLMENT VALUES (20002001,'12-AUG-16',5000,'N');

INSERT INTO INSTALLMENT VALUES (20002001,'12-SEP-16',5000,'N');

INSERT INTO INSTALLMENT VALUES (20002003,'12-JUL-16',5000,'N');

INSERT INTO INSTALLMENT VALUES (20002003,'15-SEP-16',5000,'N');

INSERT INTO INSTALLMENT VALUES (10001100,'15-OCT-16',10000,'Y');


INSERT INTO FEESPAID VALUES(20002001,'25-JUL-16','','','',5000);

INSERT INTO FEESPAID VALUES(20002001,'25-AUG-16','','','',5000);

INSERT INTO FEESPAID VALUES(20002003,'25-JUL-16','','','',5000);

INSERT INTO FEESPAID VALUES(10001100,'10-JUL-16','','','',3000);

INSERT INTO FEESPAID VALUES(20002154,'05-JUL-15','','','',24000);

INSERT INTO EXPENDITURE VALUES(100,20000,'12-SEP-16',20002,101,200,20002);

INSERT INTO EXPMASTER VALUES (101,'ADVT','CASH',9000);
```

----- create procedures or functions to solve given problem.

-- Q1> display contents of course

-- // this question is solved, so that YOU get an idea!!

```
declare
      cursor c1 is
      select *
      from course;
begin
       for z in c1 loop
            dbms_output.put_line(z.coursecode||' '||z.coursename||' '||
            z.syllabus);
      end loop;
end;
/
-- Note : turn server output on
-- set serveroutput on;
```

-- Q2> display details of all batches

-- Q3> Display details of a batch for a particular course

```
-- // this question is solved, so that YOU get an idea of passing parameters to
cursor !!

declare

      cursor c1(ccode number) is

      select *

      from batch

      where coursecode = ccode;

 begin

      for z in c1(10) loop

            dbms_output.put_line(z.batchcode||' '||z.coursecode||' '||

            z.startingdate||' '||z.coursefees||' '||z.netincome||' '||

            z.expectedincome);

      end loop;

 end;

/
```

-- Q4> Display details of a batch for a particular date. (Solve this similar to Q3). Assume some suitable batch starting date for which their is an entry in batch table.

-- Q5> Find batchcode having netincome > 5000.

--Q6> Find names of the students who enquired for the course from NAGPUR city. (Hint : use enquiry table)

--Q7> display name of the person who enquired for 'ORACLE' course. (Hint : pass courseName as parameter to cursor)

--Q8> find courseName and details of all batches. (Hint : use 2 tables course and batch. common field is courseCode)

--Q9> Define overloaded functions findenqno() to find : a> EnquiryNo if name is supplied b> EnquiryNo if refcode is supplied.

```
declare
x number;
y number;
function findenqno (fname1 varchar2) return number is  --function 1
enqno1 number (8);
begin

     -- select query here ----

     return(enqno1);
```

```
end;
function findenqno (refcode1 number) return number is  --function 2
enqno1 number (8);
begin

     -- select query here ----

     return (enqno1);
end;
begin
     -- You call the function as:
     X := findenqno ('ANIL');
     dbms_output.put_line('using name '||x);
     Y := findenqno(1001);
     dbms_output.put_line('using refcode '||y);
end;
/
```

--Q10> Define a procedure to find the average netIncome per batch. (Hint : Use aggregate function avg() and groupBy batchcode)

```
create or replace procedure batch_avgamt as
-- code here --
end batch_avgamt;
/
-- call the procedure
```

```
sql> execute batch_avgamt

--or

sql> exec batch_avgamt
```

--Q11> Define a procedure to find batches having average netIncome > 8000.(Hint : Use aggregate function avg(), groupBy batchcode and use having clause)

```
create or replace procedure batchname_avgamt_more as

-- code here --

end batchname_avgamt_more;

/
```

--Q12> Define a function to find total number of enquiries for a given advertisement. (hint: In the Enquiry table refcode is the adv code. Pass advertisementNumber as a parameter.)

```
create or replace function totenq_advcode(adv number)

return number

as

tenq number;

-- code here --

end totenq_advcode;

/
```

-- call the function as:

```
SQL> select totenq_advcode(1001) from dual;
```

--Q13> Define a procedure to print a statistics table something like this

```
/* expected output:
startamt    endAmt   no_of_students
0           1000       1
1001        5000       2
5001        10000      1
10001       15000      3
*/

create or replace procedure stat_feecollected as
tmpamt number;
i number;
inc number;
incamt number;
cnt number;
begin
      i := 0;
      tmpamt := 0;
      inc := 1000;
      while i < 20000 loop
            select count(*) into cnt
            from feespaid
            where amount between i and inc;
            dbms_output.put_line(i||'    '||inc||'    '||cnt);
            i := inc+1;
            tmpamt := tmpamt + 5000;
            inc := tmpamt;
```

```
      end loop;
end stat_feecollected;
/
```

-- call the procedure

```
sql> exec stat_feecollected;
```

--Q14> Write a procedure to calculate closing balance (or total fees collected) for each date. (hint : use sum() and groupby on feespaiddate like this : group by to_char(feespaiddate,'DD-MON-YY'))

```
create or replace procedure date_closing_amount as
```

-- code here --

```
end date_closing_amount;
/
```

```
-- call the procedure
sql> exec date_closing_amount;
```

--Q15> define a procedure to find the start and end date for a specified batch. (Hint: batchcode is passed as a parameter)

```
create or replace procedure minmaxdt_batch (bcode in number) as
```

-- code here --

```
end minmaxdt_batch;
/
```

-- call the procedure

```
sql> exec minmaxdt_batch(20002);

sql> exec minmaxdt_batch(10001);
```

--Q16> Define a View and procedure to find the month with maximum enrollments.

```
/*  step 1 : first define a View vw_enroll which holds the number of enrollments
for each date.

*/


create or replace view vw_enroll  as

select to_char(enrollmentdate) dt,count(*) cnt

from enrollment

group by to_char(enrollmentdate)

/



/* step 2 : next define a procedure which uses the above view and finds the month
name , with max enrollments. */


create or replace procedure maxenroll as

mnt varchar2(20);

ecnt number;

begin


-- code here --
```

```
end maxenroll;
/
```

-- Note: into clause should appear in select statement only once.

--Q17> Define a package for the given declaration:

```
--package specification
create or replace package test1 as
procedure add3(a in number,b in number,c in number,d out number);
function sqr(a number) return number;
end test1;
/
```

--package body

--using the package

```
declare
      ans number;
begin
      test1.add3(45,89,34,ans);
```

```
      dbms_output.put_line(ans);
      dbms_output.put_line(test1.sqr(10));
 end;
/
```

--Q18> Define a package for the given declaration :

```
--package specfication
create or replace package enq_analysis as
function finddate (enqno1 number) return date;
function finddate (fname1 varchar2,lname1 varchar2) return date;
end enq_analysis;
/
```

--package body

-- use the package

```
begin
dbms_output.put_line(enq_analysis.finddate('ANIL','SHARMA'));
dbms_output.put_line(enq_analysis.finddate(110002));
end;
```

/

--Q19> Create a function to find the name of the course having highest netIncome.

```
create or replace function crs_maxinc return varchar2 as
name varchar2(15);
begin
      select coursename
      into name
      from batch,course
      where course.coursecode = batch.coursecode
      and netincome =
      (select max(netincome)
      from batch);
      return(name);
end crs_maxinc;
/
```

-- call the function

```
select crs_maxinc from dual;
```

--Q20> Create a trigger that does not allow any DML operation on the enquiry table to be performed on any weekday, but allows insertion only on Sunday.

```
create trigger enquiry1_sunday
before insert or update or delete on enquiry
declare
      uname varchar2(20);
begin
      select user into uname from dual;
```

```
      if inserting  then
if (to_char(sysdate,'Day') != 'Sunday')
then
raise_application_error(-20111,'Insertion is not allowed on other days');
end if;
end if;
if updating or deleting then
raise_application_error(-20112,'Updation and Deletion is not  allowed ');
end if;
end;
/
```

--Q21> Create a trigger only for courseCode = 10, which allows inserting records in enquiry table, on Sunday only. No other DML allowed.

```
create trigger enquiry_type1
before insert or update or delete on enquiry
for each row
when (new.coursecode=10)
declare
      uname varchar2(20);
begin

-- code here --


end;
/
```

--Q22> Create a trigger, which generates a primary key based on the Max value of a column, of some other table.

/* it's fully solved. Student should execute and understand the effect */

-- step 1 : Create a master table and insert some values

```
create table t(a number primary key,b number);

insert into t values(1, 8);

insert into t values(5, 9);

insert into t values(7, 10);


select * from t;
```

-- step 2 : Create a trigger which ensures that the next records primary key inserted into the table is (max(primaryKey) + 1).

```
-- package specfication

create package cnt_pkg as

cnt number;

end cnt_pkg;

/


-- statement level trigger

create or replace trigger setmaxtrg before insert on t

begin

     select nvl(max(a),0) into cnt_pkg.cnt from t;
```

```
end;
/

-- row level trigger
create or replace trigger insertttrg before insert on t for each row
begin
      :new.a := cnt_pkg.cnt + 1;
      cnt_pkg.cnt := cnt_pkg.cnt + 1;
end;
/
```

--step 3:

```
-- perform an insert on t , this will invoke the trigger.
insert into t values(2,30);

-- check the new record inserted.
select * from t;
-- why is new record values (8,30) instead of (2,30);
-- Because of the trigger
```

--Q23> Create a trigger , to implement update cascade functionality. i.e. if a update a parent row , corresponding foreign key columns in the child rows are also updated.

--step 1 : create parent table and insert few records

```
create table parent(a int primary key, b int);
insert into parent values(1,2);
```

```
insert into parent values(2,3);

insert into parent values(3,5);

--step 2 : create child table and insert few records

create table child(a int references parent(a), c int);

insert into child values(1,10);

insert into child values(1,11);

insert into child values(3,30);

insert into child values(2,14);


insert into child values(4,1); -- integrity constraint violation


commit;


--step 3 : create trigger

create trigger update_cascade after update on parent for each row

begin

update child set a = :new.a where a = :old.a;

end;

/


-- now test the trigger :

update parent set a = 5 where a = 1 ;

select * from parent;

select * from child;
```

--Q24> Create a trigger to reverse update. i.e. if we update a child row, the corresponding primary key of the parent row should be updated.

-- use the same parent and child tables as created above.

```
create or replace trigger rev_update_cascade before update on child

for each row

begin

update parent set a = :new.a where a = :old.a;

end;

/


select * from parent;

select * from child;


update child set a = 10 where a = 2;
```

-- Note : we would get mutating error.

-- what is Mutating error ?

```
/*

Your trigger is AFTER or before UPDATE OR INSERT OR DELETE. Means if you run
UPDATE OR INSERT OR DELETE statements on this table, the trigger will fire. But
you are trying to update the same table again inside your trigger, which is
wrong!!! This is why you are getting the error. You cannot modify the same table,
the trigger is firing on.

Remember the parent-child tables are related.

*/
```

-- This example is given to see and understand Mutating error.