# Assignment 1a

**SUPPLIER (SID, SNAME, CITY)**

**PART (PID, PNAME, COLOR)**

**CATALOG (SID, PID, COST)**

1. Create the above tables with necessary constraints.

CREATE TABLE SUPPLIER (

SID INT PRIMARY KEY,

SNAME VARCHAR2(15),

CITY VARCHAR2(10)

);

CREATE TABLE PART (

PID INT PRIMARY KEY,

PNAME VARCHAR2(15),

COLOR VARCHAR2(10)

);

CREATE TABLE CATALOG (

SID INT,

PID INT,

COST INT,

FOREIGN KEY(SID) REFERENCES SUPPLIER(SID),

FOREIGN KEY(PID) REFERENCES PART(PID)

);

1. Display details of suppliers who supply red colored parts.

SELECT \* FROM SUPPLIER

WHERE SID IN (SELECT SID FROM CATALOG

WHERE PID IN (SELECT PID FROM PART WHERE COLOR='RED'));

1. Display total number of suppliers for each part.

SELECT PID, COUNT(SID) AS SP\_NO FROM CATALOG

GROUP BY PID ORDER BY PID;

1. Display suppliers based in Mumbai and whose names have ‘j’ as the third character.

SELECT \* FROM SUPPLIER

WHERE CITY='MUMBAI' AND SNAME LIKE '\_\_J%';

1. Create a view ‘catview’ which stores details about suppliers and parts where the cost is more than 100.

CREATE VIEW CATVIEW AS

(SELECT SID,PID FROM CATALOG WHERE COST>100);

1. Add a column SCONTACT to the table SUPPLIER which always has ‘0’(zero) as the first characters.

ALTER TABLE SUPPLIER

ADD SCONTACT VARCHAR2(11) CONSTRAINT SCONTACT\_CHECK CHECK (SCONTACT LIKE '0%');

1. Create a composite index on table SUPPLIER for the columns SNAME and CITY.

CREATE INDEX idx\_supplier\_name\_city ON SUPPLIER (SNAME, CITY);

1. Display all those suppliers who supply more than three parts.

SELECT \* FROM SUPPLIER

WHERE SID IN (SELECT SID FROM CATALOG GROUP BY SID HAVING COUNT(PID)>3);

1. Display suppliers who are not supplying any part.

SELECT \* FROM SUPPLIER S WHERE NOT EXISTS(SELECT 1 FROM CATALOG WHERE S.SID=CATALOG.SID) ORDER BY SID;

1. Display the average cost of all green parts.

SELECT AVG(COST) GREEN\_COST FROM CATALOG WHERE PID IN (SELECT PID FROM PART WHERE COLOR='GREEN');

**CATEGORY (CAT\_NO, CAT\_TYPE, EXPIRY\_DT)**

**ITEM (ITEMNO, IDESC, UNIT\_OF\_MEASURE, QTY\_ON\_HAND, WEIGHT, CAT\_NO)**

1. Create the above tables with necessary constraints.

CREATE TABLE CATEGORY(

CAT\_NO INT PRIMARY KEY,

CAT\_TYPE VARCHAR2(15) NOT NULL,

EXPIRY\_DT DATE

);

CREATE TABLE ITEM (

ITEM\_NO INT PRIMARY KEY,

IDESC VARCHAR2(20),

UNIT\_OF\_MEASURE VARCHAR2(7),

QTY\_ON\_HAND INT,

WEIGHT NUMBER(6,2),

CAT\_NO INT,

FOREIGN KEY(CAT\_NO) REFERENCES CATEGORY(CAT\_NO)

);

1. Add a constraint to the ITEM table which will allow the QTY\_ON\_HAND to have values between 100 and 1000 only.

ALTER TABLE ITEM

ADD CONSTRAINT QTY\_CHECK CHECK(QTY\_ON\_HAND BETWEEN 100 AND 1000);

1. Display how many items are there in each category.

SELECT COUNT(\*), CAT\_NO FROM CATEGORY C LEFT JOIN ITEM I ON C.CAT\_NO=I.CAT\_NO ORDER BY CAT\_NO;

OR

SELECT COUNT(\*),CAT\_NO FROM ITEM GROUP BY CAT\_NO ORDER BY CAT\_NO;

1. Display the items in descending order of their weights.

SELECT \* FROM ITEM ORDER BY WEIGHTS DESC;

1. Display categories which expire in the month of December.

SELECT \* FROM CATEGORY WHERE TO\_CHAR(EXPIRY\_DT, ’MM’) =12;

1. Display items which have the same weight as that of ITEMNO 116.

SELECT \* FROM ITEM

WHERE WEIGHT = (SELECT WEIGHT FROM ITEM WHERE ITEM\_NO=116);

1. Display items which do not have the alphabet ‘A’ or ‘T’ in their IDESC.  
   SELECT \* FROM ITEM WHERE IDESC NOT LIKE ‘%A%’ AND IDESC NOT LIKE ‘%T%’;
2. Increase the size of the column IDESC by making it a variable length of 25 characters.

ALTER TABLE ITEM

MODIFY COLUMN IDESC VARCHAR2(25);

1. Display the item details and the category details of items which have a UNIT\_OF \_MEASURE as Kilogram, Meters or Litres.

SELECT \* FROM CATEGORY C JOIN ITEM I ON C.CAT\_NO = I.CAT\_NO WHERE I.UNIT\_OF\_MEASURE IN (‘Kilogram’,’Meters’,’LItres’)

# Assignment 1b

**TEAM\_MASTER (TEAM\_ID, TEAM\_NAME)**

**PLAYER (TEAM\_ID, PID, PNAME, PBIRTHDATE)**

**SCORE (TEAM\_ID, PID, RUNS\_MADE, OUT\_TYPE, B\_ID, BTEM\_ID)**

**BALLING (TEAM\_ID, B\_ID, OVERS, MAIDEN, RUNS\_GIVEN, WICKET\_TAKEN)**

In the above tables, B\_ID is the bowler id and refers to the P\_ID of PLAYER table. The SCORE table stores details about the scores of the batsmen and also stores details about how t hey got out. The BALLING table stores details about the bowlers.

1. Write DDLs to create the above table with necessary constraints.

CREATE TABLE TEAM\_MASTER(

TEAM\_ID INT PRIMARY KEY,

TEAM\_NAME VARCHAR2(10)

);

CREATE TABLE PLAYER(

TEAM\_ID INT,

PID INT,

PNAME VARCHAR2(15),

PBIRTHDATE DATE,

PRIMARY KEY(TEAM\_ID,PID),

FOREIGN KEY(TEAM\_ID) REFERENCES TEAM\_MASTER(TEAM\_ID)

);

CREATE TABLE SCORE(

BTEAM\_ID INT,

TEAM\_ID INT,

PID INT,

RUNS\_MADE INT,

OUT\_TYPE VARCHAR2(10),

B\_ID INT,

PRIMARY KEY(BTEAM\_ID),

FOREIGN KEY(TEAM\_ID, PID) REFERENCES PLAYER(TEAM\_ID, PID),

FOREIGN KEY(B\_ID) REFERENCES BALLING(B\_ID)

);

CREATE TABLE BALLLING(

B\_ID INT,

TEAM\_ID INT,

OVERS INT,

MAIDEN INT,

RUNS\_GIVEN INT,

WICKETS\_TAKEN INT,

PRIMARY KEY (B\_ID),

FOREIGN KEY(B\_ID, TEAM\_ID) REFERENCES PLAYER (PID,TEAM\_ID)

);

1. Display details about the bowler who took the maximum wickets.

SELECT \* FROM BALLING WHERE WICKET\_TAKEN=(SELECT MAX(WICKET\_TAKEN) FROM BALLING);

1. Which players are born in the month of October and have scored more than 25 runs?

SELECT \* FROM PLAYER WHERE EXTRACT(MONTH FROM PBIRTHDATE)=10 AND PID IN(SELECT PID FROM SCORE WHERE RUNS\_MADE>25);

1. Display details about players who bowled at the most three maiden overs.

SELECT \* FROM PLAYER WHERE PID IN (SELECT B\_ID FROM WHERE MAIDEN <4);

or

SELECT \* FROM PLAYER P JOIN BALLING B ON P.PID=B.B\_ID WHERE B.MAIDEN<4;

1. Display the total runs given by each bowler.

SELECT B.B\_ID, B.RUNS\_GIVEN FROM BALLING B JOIN PLAYER P ON B.B\_ID=P.PID;

1. Display team wise how many players got out with lbw, stumping, catch etc.

SELECT TEAM\_ID, OUT\_TYPE, COUNT(\*) TOTAL FROM SCORE GROUP BY (TEAM\_ID, OUT\_TYPE) ORDER BY TEAM\_ID;

1. Display details about players who are more than 25 years old.

SELECT \* FROM PLAYER

WHERE TRUNC(MONTHS\_BETWEEN(SYSDATE, PBIRTHDATE) / 12) > 25;

1. Display the bowlers who are also batsmen.

SELECT \* FROM PLAYER WHERE PID IN (SELECT PID FROM SCORE S JOIN BALLING B ON S.PID = B.B\_ID);

1. Create a read only view, which contains details about the team name and all its players.

CREATE VIEW TEAM\_DET AS SELECT \* FROM TEAM\_MASTER T JOIN PLAYER P ON T.TEAM\_ID=P.TEAM\_ID;

**CANDIDATE(CID,CNAME,CADDRESS, CBIRTH\_DT)**

**TEST(TID,TNAME,TOT\_MARKS,PASS\_MKS)**

**TEST\_CENTRE(TCID,LOCATION,MGR,CAPACITY)**

**TEST\_TAKEN(CID,TID,TCID,TEST\_DT,SCORE)**

1. Design the above tables with appropriate constraints.

CREATE TABLE CANDIDATE(

CID INT PRIMARY KEY,

CNAME VARCHAR2(20),

CADDRESS VARCHAR2(35),

CBIRTH\_DT DATE

)

CREATE TABLE TEST(

TID INT PRIMARY KEY,

TNAME VARCHAR2(15),

TOT\_MARKS INT,

PASS\_MKS INT

)

CREATE TABLE TEST\_CENTRE(

TCID INT PRIMARY KEY,

LOCATION VARCHAR2(25),

MGR VARCHAR,

CAPACITY INT

)

CREATE TABLE TEST\_TAKEN(

CID INT,

TID INT,

TCID INT,

TEST\_DT DATE,

SCORE FLOAT,

PRIMARY KEY(CID,TID,TCID),

FOREIGN KEY(TCID) REFERENCES TEST\_CENTRE(TCID),

FOREIGN KEY(TID) REFERENCES TEST(TID),

FOREIGN KEY(CID) REFERENCES CANDIDATE(CID)

)

1. Display the number of candidates who have appeared for each test at each test centre.

SELECT TID, TCID, COUNT(\*) FROM TEST\_TAKEN GROUP BY (TCID,TID) ORDER BY TCID;

1. Display candidate details for those candidates who have scored as many marks as Peter for tests conducted at Bangalore.

SELECT \* FROM CANDIDATE WHERE

CID IN (SELECT CID FROM TEST\_TAKEN

WHERE SCORE=(SELECT SCORE FROM TEST\_TAKEN

WHERE CID=(SELECT CID FROM CANDIDATE

WHERE CNAME='PETER')) AND

TCID=(SELECT TCID FROM TEST\_CENTRE WHERE LOCATION='BANGALORE'))

1. Display details about test centers where no tests have been conducted.

SELECT \* FROM TEST\_CENTRE TC WHERE NOT EXISTS(SELECT 1 FROM TEST\_TAKEN WHERE TC.TCID=TEST\_TAKEN.TCID);

1. Display details of candidates who have failed.

SELECT \* FROM CANDIDATE C JOIN TEST\_TAKEN TT ON C.CID=TT.CID JOIN TEST T ON TT.TID =T.TID WHERE SCORE<PASS\_MKS;

1. For tests, which have been conducted between 2-3-04 and 23-4-04, show details of the tests as well as the test centers.

SELECT T.TID, T.TNAME, T.TOT\_MARKS, T.PASS\_MKS, TC.TCID, TC.LOCATION, TC.MGR, TC.CAPACITY , TC.TEST\_DT FROM TEST T JOIN TEST\_TAKEN TT ON T.TID=TT.TID JOIN TEST\_CENTRE TC ON TT.TCID = TC.TCID WHERE TT.TEST\_DT BETWEEN '02-MAR-04' AND '23-APR-04'

1. How many candidates appeared in the ‘ORACLE FUNDAMENTALS’ test at Surat in the month of February?

SELECT COUNT(\*) FROM TEST\_TAKEN WHERE TID IN(SELECT TID FROM TEST WHERE TNAME='ORACLE') AND TCID IN (SELECT TCID FROM TEST\_CENTRE WHERE LOCATION='SURAT') AND EXTRACT(MONTH FROM TEST\_DT)=2

1. Display details about candidates who appeared for tests in the same month as the month in which they were born.

SELECT \* FROM CANDIDATE C JOIN TEST\_TAKEN T ON C.CID=T.CID

WHERE EXTRACT(MONTH FROM C.CBIRTH\_DT)= EXTRACT( MONTH FROM T.TEST\_DT);

1. Display the details about candidates who have scored the highest in each test, test centre wise.

SELECT TID, TCID, MAX(SCORE ) FROM TEST\_TAKEN GROUP BY TCID,TID ORDER BY TCID;

OR

SELECT C.CID, C.CNAME, T.TID, T.TNAME, TC.TCID, TC.LOCATION FROM CANDIDATE C JOIN TEST\_TAKEN TT

ON C.CID=TT.CID JOIN TEST T

ON TT.TID=T.TID JOIN TEST\_CENTRE TC

ON TT.TCID=TC.TCID

WHERE (TT.TID,TT.TCID,TT.SCORE) IN

(SELECT TID,TCID,MAX(SCORE) FROM TEST\_TAKEN GROUP BY TCID,TID)

ORDER BY TT.TCID,TID ;

1. Design a read only view that has details about candidates and the tests that he has appeared for.

CREATE VIEW CANDIDATE\_TEST AS SELECT \* FROM CANDIDATE C JOIN TEST\_TAKEN TT ON C.CID = TT.CID WITH READ ONLY;

# Assignment 2

1. Write a PL/SQL block which finds the first 10 prime numbers.

declare

n number :=2;

isPrime Boolean;

c number(2):=0;

begin

loop

exit when c=10;

isPrime:=TRUE;

for i in 2..trunc(sqrt(n)) loop

if mod(n,i)=0 then

isPrime:=FALSE;

exit;

end if;

end loop;

if isPrime then

c:=c+1;

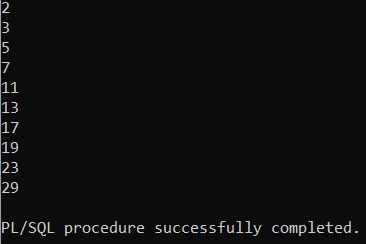
dbms\_output.put\_line(n);

end if;

n:=n+1;

end loop;

end;



1. Write a PL/SQL block to find if the given string is a palindrome. (spelt the same from both ends e.g. MADAM, RADAR, LEVEL, NOON, etc.)

declare

str varchar2(100);

revStr varchar2(100);

begin

str:=&str;

revStr:=reverse(str);

if str=revStr then

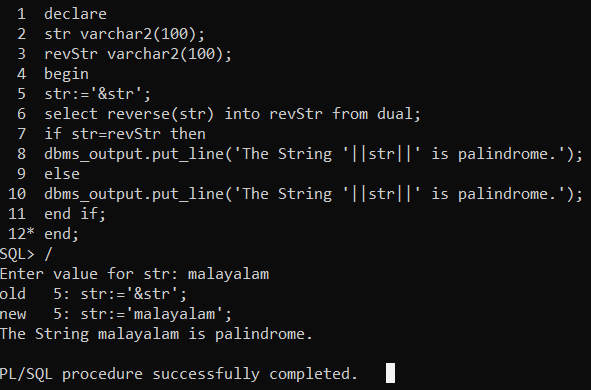
dbms\_output.put\_line(‘The String ’||str||’ is palindrome.’);

else

dbms\_output.put\_line(‘The String ’||str||’ is palindrome.’);

end if;

end;



1. Take as input the employee’s name from the user. If this employee is either a MANAGER ANALYST gives him a raise of 15% else decrease his salary by 12%. Write appropriate exception handlers.

declare

eName varchar2(100);

eDesignation varchar2(100);

eSal number;

begin

eName := ‘&Employee Name’;

select salary into eSal, designation into eDesignation from emp where name = eName;

--make sure there is only one record of name otherwise code will break

if eDesignation in (‘MANAGER’, ‘ANALYST’) then

eSal := eSal\*1.15;

else

eSal := eSal\*0.88;

end if;

update emp

set sal= eSal

where name = eName

commit;

dbms\_output.put\_line(‘Salary updated successfully.’);

exception

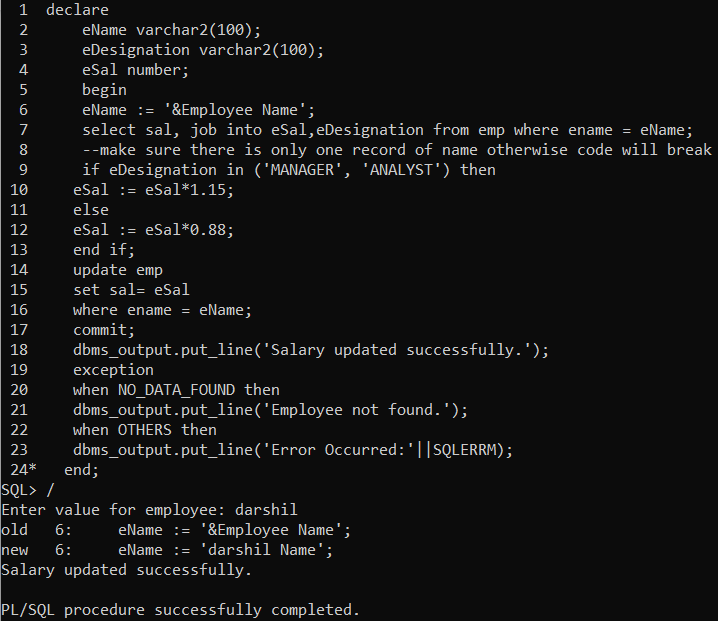
when NO\_DATA\_FOUND then

dbms\_output.put\_line(‘Employee not found.’);

when OTHERS then

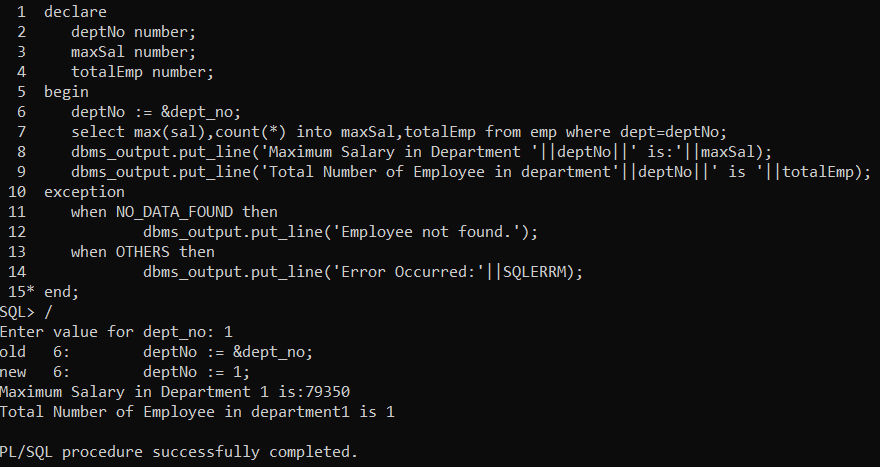
dbms\_output.put\_line(‘Error Occurred:’||SQLERRM);

end;



1. Take as input department number from the user and display the maximum salary and total number of employees in that department.

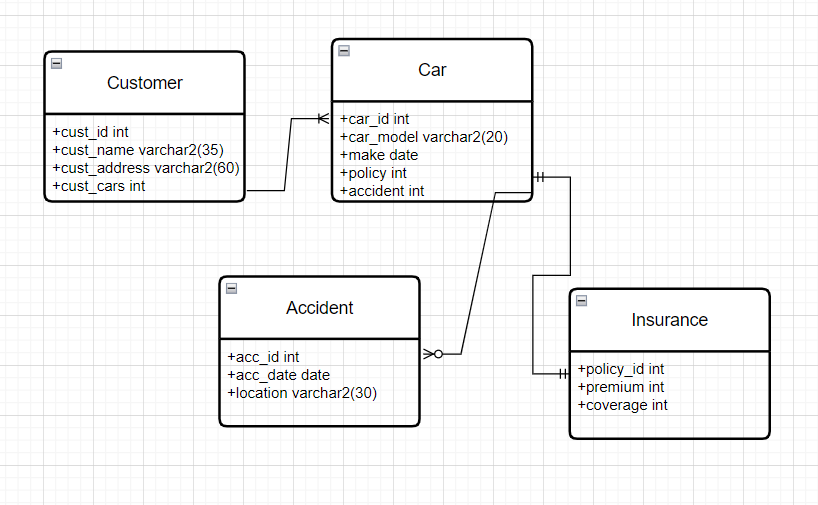
Write appropriate exception handlers.



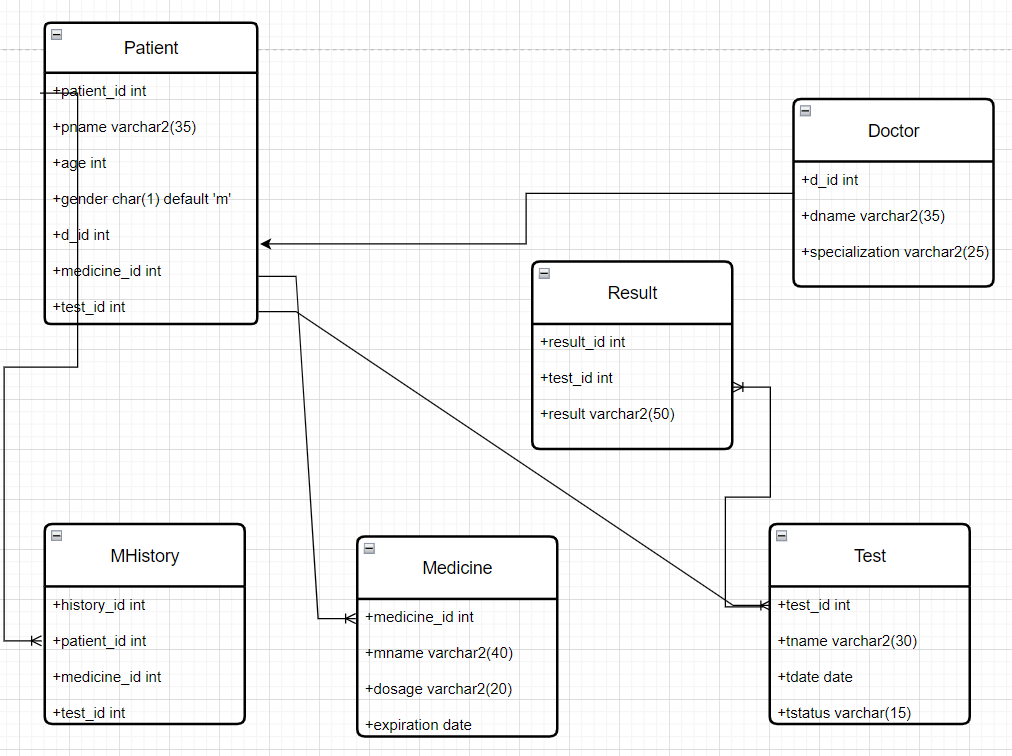
# Assignment 3

# E-R Diagram

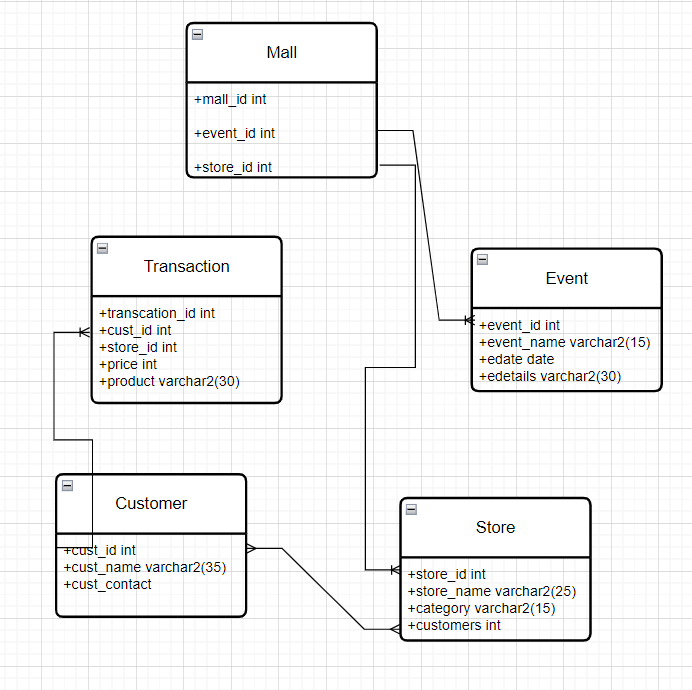
1. Construct an E-R Diagram for a car insurance company with a set of customers, each of whom owns a number of cars. Each car has a number of recorded accidents associated with it. The details about the insurance policies of the cars as well as the details of the accidents are also stored.



1. Construct an E-R Diagram for a hospital management system. The details of the patients and the doctors will be required as well as the details of the tests and its results conducted on a patient. The patient history is also stored for future reference. Everything related to the medicines prescribed is also required.



1. Draw an E-R diagram to depict the different activities that go on in a shopping mall. Take Vadodara Central as a case study and try to design your tables if you were to computerize its activities. Mention your scope very clearly.



# Assignment 4

1. Write PL/SQL block which will give a raise in salary to employees as per the following:

if sal+comm < 5000 increase by 10% of sal

if sal+comm >= 5000 increase by 500 + 12% of sal above 7000

Insert the raise details in the table bonus (empno, sal, bonus\_amt)

declare

v\_bonus\_amt number;

cursor c1 is select empno, sal, nvl(comm, 0) as comm from emp;

begin

for r1 in c1 loop

if (r1.sal + r1.comm) < 5000 then

v\_bonus\_amt := r1.sal \* 0.10;

elsif (r1.sal + r1.comm) >= 5000 then

v\_bonus\_amt := 500 + ((r1.sal - 7000) \* 0.12);

end if;

insert into bonus (empno, sal, bonus\_amt)

values (r1.empno, r1.sal, v\_bonus\_amt);

end loop;

commit;

end;

2. Suppose you have created the following three tables in your database :

PART\_MASTER (P#,PNAME, PRICE, TOT\_QTY, REORDER\_LEVEL)

PART\_TRANS (O#, P#, QTY\_REQ), where O# is the order no.

TEMP (P#,QTY)

Write a PL/SQL block which will read records one by one from the transaction table and will check whether after issuing this part, the total quantity of this part is going below the reorder level in the master table or not. If it is then that part is not to be issued, hence give appropriate message and enter the details of this part in the TEMP table. If the total quantity after issuing that part is not going below the reorder level then modify the master table accordingly.

declare

tempQty number;

cursor c1 is select \* from part\_trans;

begin

for r1 in c1 loop

select tot\_qty into tempQty from part\_master where p# = r1.p#;

if tempQty - r1.qty\_req < r1.reorder\_level then

insert into temp (p#, qty) values (r1.p#, r1.qty\_req);

dbms\_output.put\_line('part ' || r1.p# || ' cannot be issued as it will fall below reorder level.');

else

update part\_master

set tot\_qty = tot\_qty - r1.qty\_req

where p# = r1.p#;

dbms\_output.put\_line('part ' || r1.p# || ' issued successfully.');

end if;

end loop;

commit;

end;

3. Given a table ISSUE(ROLLNO,BOOKNO,ISSUE\_DATE,RETURN\_DATE),

check which students have to pay fine and the amount of fine to be paid. A student can keep the book for fifteen days. If the number of days has exceeded 15 then fine is calculated as follows:

Upto 7 days 50 paise per day

8 – 15 Re. 1 per day from 8th day onwards

more than 15 days Rs. 1.5 per day from 16th day onwards

After calculating the fine store the required information in the FINE table so that a report can later on printed out.

declare

fine number;

cursor c1 is select \* from issue;

begin

for r1 in c1 loop

if round(r1.return\_date-r1.issue\_date)<=15 then

fine:=0;

elsif round(r1.return\_date - r1.issue\_date) <= 22 and round(r1.return\_date - r1.issue\_date)>15 then

fine:=0.50 \* round(r1.return\_date - r1.issue\_date);

elsif round(r1.return\_date - r1.issue\_date)> 22 and round(r1.return\_date - r1.issue\_date) <= 30 then

fine:=1 \* round(r1.return\_date - r1.issue\_date);

else

fine:=1.5 \* round(r1.return\_date - r1.issue\_date);

end if;

if fine > 0 then

insert into fine (rollno, bookno, fine\_amount) values (r1.rollno, r1.bookno, fine);

end if;

end loop;

commit;

end;

# Assignment 5

* 1. Write a PL/SQL block for the following table as per the requirement :

CANDIDATE (SEAT\_NO, FORM\_NO, NAME, CATEGORY, PERCENTAGE)

Except the seat numbers, all other information regarding the candidates appearing in the MCA entrance test is available in the CANDIDATE table. Seat numbers are to be generated as per the following:

Seat numbers are 8 characters in length. The first three characters are ‘111’. The fourth character represents the category to which the candidate belongs. The rest of the four characters are used to denote the rank of the candidate as per his percentage, category-wise. For e.g., a candidate belonging to the GENERAL category and having secured the highest percentage amongst all other candidates in the same category will have a seat number as ‘11120001’, where ‘2’ is the code for the GENERAL category. In this way generate the seat numbers for all the candidates in the CANDIDATE table.

Please mention any assumptions that you make. The codes for the different categories are:

GENERAL : 2

SC : 3

ST : 4

SEBC : 5

PH : 6

declare

catgoryCode number;

seatNo varchar2(8);

cursor c1 is select \* from candidate;

begin

for r1 in c1

loop

case r1.category

when 'GENERAL' then catgoryCode := 2;

when SC then catgoryCode := 3;

when ST then catgoryCode := 4;

when ‘SEBC’ then catgoryCode := 5;

when 'PH' then catgoryCode := 6;

end case;

seatNo := '111' || to\_char(catgoryCode) || lpad(r1.rownum, 4, '0');

update candidate

set seat\_no = seatNo

where seat\_no is null and rowid = r1.rowid;

commit;

end loop;

end;

* 1. For the table STUDENT( Rollno, Name, Mrks1, Mrks2, Mrks3, Percentage), write a PL/SQL code which calculates the percentage for each student assuming passing marks as 40 and total marks as 100 for each subject. Also display the details about the students who have scored the highest in each subject.

declare

cursor c1 is select \* from student;

sdata student%rowtype;

begin

for s in c1

loop

update student set percentage= ((s.mrks1 + s.mrks2 + s.mrks3) / 3)

where current of c1;

end loop;

select \* into sdata from student where mrks1=(select max(mrks1) from student);

dbms\_output.put\_line('highest marks in mrks1: ' || sdata.name);

select \* into sdata from student where mrks2=(select max(mrks2) from student);

dbms\_output.put\_line('highest marks in mrks2 ' ||sdata.name);

select \* into sdata from student where mrks3=(select max(mrks3) from student);

dbms\_output.put\_line('highest marks in mrks3: ' ||sdata.name);

end;

/

For the following tables,

CANDIDATE (CID, CNAME, CADDRESS, CBIRTH\_DT)

TEST (TID, TNAME, TOT\_MRKS, PASS\_MKS)

TEST\_CENTRE (TCID, LOCATION, MGR, CAPACITY)

TEST\_TAKEN (CID, TID, TCID, TEST\_DT, SCORE)

1. Write a PL/SQL block which will accept the test id and test center id from the user and display details about all those candidates who have scored more than average for that test. Also display details about the dates on which that test center had full attendance.

declare

test\_id number;

test\_center\_id number;

average\_score number;

cursor c1(testID,avgScore) is select \* from test\_taken where tid=testID and score>avgScore;

cursor c2(testCenterId) is select test\_dt from test\_taken where tcid = testCenterId group by test\_dt having count(\*) = (select capacity from test\_centre where tcid = testCenterId))

begin

test\_id := &test\_id;

test\_center\_id := &test\_center\_id;

select avg(score) into average\_score from test\_taken where tid = test\_id;

for c in c1(test\_id,average\_score)

loop

dbms\_output.put\_line('candidate id: ' || c.cid || ', score: ' || c.score);

end loop;

for d in c2(test\_center\_id)(

loop

dbms\_output.put\_line('test center full attendance date: ' || to\_char(d.test\_dt, 'dd-mon-yyyy'));

end loop;

end;

1. Write a PL/SQL block, which will accept the candidate id from the user and for this user display details about all the tests that he has appeared for as well as the details about his scores and the maximum scores of those tests.

declare

candidate\_id number;

cursor c1(candidateId) is select tt.tid, tt.test\_dt, tt.score, tst.tot\_mrks from test\_taken tt join test tst on tt.tid = tst.tid where tt.cid = candidateId;

begin

candidate\_id := &candidate\_id;

for t in c1(candidate\_id)(

)

loop

dbms\_output.put\_line('test id: ' || t.tid || ', test date: ' || to\_char(t.test\_dt, 'dd-mon-yyyy') || ', score: ' || t.score || ', maximum marks: ' || t.tot\_mrks);

end loop;

end;

# Assignment 6

* + 1. Write a PL/SQL main program and two functions which will do the following:

In the main program take as input the deptno and job. Pass the deptno to a function ‘findloc’.

This function will find the location of the department and send it back to the main program. Check whether the location is Mumbai or Chennai. If it is, call another function ‘findtotjob’ which will take as input the location and job and send back the total number of employees working at that location with that particular job title back to the main program.

Display appropriate messages and write the needful exceptions also.

create or replace function findloc( dno number)

return varchar2 is

loc varchar2(100);

begin

select location into loc from dept where department\_id = dno ;

return loc;

exception

when NO\_DATA\_FOUND then

return null;

end findloc;

create or replace function findtotjob( loc in varchar2,job in varchar2)

return number is

total number;

begin

select count(\*) into total from emp e join dept d on e.deptno = d.dno

where d.location = loc and e.job = job;

return total;

exception

when NO\_DATA\_FOUND then

return null;

end findtotjob;

declare

deptno dept.dno%type := &deptno;

job emp.job\_id%type := '&job';

loc dept.location%type;

totalEmp number;

begin

loc := findloc(v\_deptno);

if loc = 'mumbai' or loc = 'chennai' then

totalEmp := findtotjob(loc, job);

if totalEmp is not null then

dbms\_output.put\_line('total employees in ' || loc || ' with job ' || job || ': ' || totalEmp);

else

dbms\_output.put\_line('no employees found in ' || loc || ' with job ' || job);

end if;

else

* 1. Write a PL/SQL block that accepts an employee number from the user.

The salary, commission, department number and job of this employee are passed to a procedure ‘p1’. In the procedure check whether the salary is more than 10% of the commission and whether the employee is a manager. If both the conditions are true, increase the salary by 25% and send this new value back to the calling program. Also send the department name of department where this employee is working to the calling program. Update the record in the main program accordingly.

Display appropriate messages and write the needful exceptions also.

create or replace procedure p1(edata emp%rowtype,salary out number, deptName out varchar2) is

begin

if edata.salary > (0.1 \* edata.comm) and edata.job = 'manager' then

update emp

set salary = sal \* 1.25

where empno = edata.empno;

select dname into deptName from dept where deptno = edata.dno;

else

salary := edata.salary;

p\_dept\_name := null;

end if;

exception

when NO\_DATA\_FOUND then

dbms\_output.put\_line('employee not found.');

when OTHERS then

dbms\_output.put\_line('an error occurred: ' || SQLERRM);

end;

/

* 1. Write a PL/SQL package which has the following objects:

A procedure ‘p2’ which accepts an employee number and a department number from the main program. It calls a function ‘f2’ with these parameters and the function returns ‘true’ if the employee is working in that department otherwise returns ‘false’. In case the value is ‘true’, the procedure returns the total number of employees working in that department otherwise returns null. The package also has a constant ‘title’ which has the value ‘Total Employees’. Use this constant in the procedure / calling program while displaying the requisite message.

Write a calling program which uses this package.

create or replace package employee\_package as

procedure p2(eNo number, dNo number, total out number);

function f2(eNo number, dNo number) return Boolean;

title constant varchar2(100) := 'Total Employees';

end employee\_package;

/

create or replace package body employee\_package as

procedure p2( eNo number, dNo number, total out number ) is

begin

if f2(eNo,dNo) then

select count(\*) into total from emp where deptno = dNo;

else

total:= null;

end if;

exception

when NO\_DATA\_FOUND then

total := null;

when OTHERS then

dbms\_output.put\_line('an error occurred: ' || SQLERRM);

end p2;

function f2(eNo number, dNo number) return Boolean is

result Boolean;

begin

select 1 into result from employees

where empno = eNo and deptno =dNo;

return TRUE;

exception

when NO\_DATA\_FOUND then

return FALSE;

when OTHERS then

dbms\_output.put\_line('an error occurred: ' || SQLERRM);

return FALSE;

end f2;

end employee\_package;

declare

eNo emp.empno%type := &eNo;

dNo emp.deptno%type := &dNo;

totalEmp number;

begin

employee\_package.p2(eNo, dNo, totalEmp);

if totalEmp is not null then

dbms\_output.put\_line(employee\_package.title || ' in department ' || dNo || ': ' || totalEmp);

else

dbms\_output.put\_line('no employees found in department ' || dNo);

end if;

exception

when OTHERS then

dbms\_output.put\_line('an error occurred: ' || SQLERRM);

end;

# Assignment 7

For the following relations, write relational algebraic solutions as per the requirements:

**STAFF (EMPNO, NAME, SAL, AGE, NUMPUBL, DNO)**

**DEPT (DNO, DNAME, MANAGER, FLOOR)**

* + 1. Retrieve information about the CS department.

π(EMPNO, NAME, SAL, AGE, NUMPUBL, DNO, MANAGER)(σ(DNAME='CS')(STAFF ⨝ DEPT))

* + 1. Retrieve name and salary about faculty whose age is greater than 30 or whose number of publications is greater than 10.

π(NAME, SAL)(σ(AGE > 30 ∨ NUMPUBL > 10)(STAFF))

* + 1. Retrieve names of staff members who work in departments located on the fifth floor.

π(NAME)(σ(FLOOR = 5)(STAFF ⨝ DEPT))

* 1. Retrieve names of staff members who work for all the departments.

π(NAME)((STAFF ⨝ DEPT) ÷ DNO)

* 1. Retrieve names of managers who have not published any papers.

π(MANAGER)(STAFF ⨝ DEPT - (σ(NUMPUBL > 0)(STAFF ⨝ DEPT)))

OR

π(MANAGER) σ(MANAGER=EMPNO ^ NUMPUBL = 0)(STAFF ⨝ DEPT)

* 1. Retrieve the total number of staff members working in each department.

DNO, DNO *G* COUNT(\*) (STAFF ⨝ DEPT)

* 1. Update the staff table, increasing the salary of all those members who have publications by 20% and decreasing the salaries of the rest by 5%.

STAFF ← π EMPNO, NAME, SAL\*1.2, AGE, NUMPUBL, DNO(STAFF ⨝ π EMPNO (σ NUMPUBL > 0 (STAFF))

STAFF ← π EMPNO, NAME, SAL\*0.95, AGE, NUMPUBL, DNO( STAFF ⨝ π EMPNO (σ NUMPUBL = 0 (STAFF))

* 1. Insert a new record in the DEPT table.

DEPT ← DEPT ∪ {(5,’CRM’,123,6) }

* 1. Delete all those staff members who are either working on the second floor or are managers.

STAFF ← STAFF - (π(EMPNO, NAME, SAL, AGE, NUMPUBL, DNO)(σ(FLOOR = 2 ∨ EMPNO IN (SELECT MANAGER FROM DEPT))(STAFF ⨝ DEPT)))

* 1. Retrieve the details of staff members and the details of departments they are working in if they are managers.

π(NAME, SAL, AGE, NUMPUBL, STAFF.DNO, DNAME)( σ(MANAGER = EMPNO)(STAFF ⨝ DEPT))

1. Retrieve details of employees and their departments as well as details of departments where no employees are working.

π(EMPNO, NAME, SAL, AGE, NUMPUBL, DNO, DNAME, MANAGER, FLOOR)((STAFF ⨝ DEPT) ∪ (σ(deptno)(dept) – σ (deptno)(emp)))

# Assignment 8

1. For the following tables:

**STUDENT (SEATNO, NAME, MARKS1, MARKS2, MARKS3)**

**RESULT (SEATNO, AVERAGE)**

* + - 1. Write a database trigger which is fired every time a record is inserted in the student table. In the trigger the average of the best two of the three marks is calculated and inserted in the result table.

create or replace trigger calc\_avg\_trigger

after insert on student

for each row

begin

insert into result

values (:NEW.seatno, (select avg(marks) from (

select \* from (select marks1 as marks from dual

union all

select marks2 from dual

union all

select marks3 from dual)

order by marks desc) where rownum <= 2));

end;

* + - 1. Write a database trigger which adds an ‘I’ before every SEATNO value when a new record is being inserted if an ‘I’ is not already present. The SEATNO should not be more than five characters in length and last character should not be numeric. Also see that if the student is ‘Tom’, ‘Dick’, or ‘Harry’ then no DML is to be allowed against this table.

create or replace trigger insert\_student\_trigger

before insert on student

for each row

declare

sNo varchar2(5);

begin

-- check if the student is 'tom', 'dick', or 'harry'

if :NEW.name in ('tom', 'dick', 'harry') then

raise\_application\_error(-20001, 'inserts for tom, dick, or harry are not allowed.');

end if;

if length(sNo) > 5 or regexp\_like(substr(sNo, -1), '[0-9]') then

raise\_application\_error(-20002, 'invalid seatno format.');

end if;

sNo := :NEW.seatno;

if length(sNo) < 5 then

if substr(sNo, 1, 1) <> 'i' then

sNo := 'i' || lpad(sNo,4,’0’);

end if;

else

if substr(sNo,1,1<>’I’ then

raise\_application\_error(-20002, 'invalid seatno format.');

end if;

end if;

-- assign the modified seatno back to :new

:new.seatno := sNo;

end;

/

1. For the following tables:

**MASTER (MACCNO, NAME, BALANCE)**

**TRANS (TACCNO, TNAME, TYPE, AMOUNT)**

**AUDIT (ACCNO, TYPE, USER\_ID, SYS\_DATE, AMOUNT)**

TYPE column contains information regarding withdrawal or deposit.

Write a database trigger that is fired against every DML on the TRANS table and will do the following:

* + - 1. If corresponding record exists in the MASTER table, update it.
      2. If corresponding record does not exist, insert it as a new record in MASTER.
      3. For every action taken, note the necessary details in the AUDIT table.

create or replace trigger trans\_trigger

after insert or update on trans

for each row

declare

bal number;

begin

if inserting or updating then

select balance into bal from master

where maccno = :new.taccno for update;

if sql%found then

if :new.type = 'withdrawal' then

update master

set balance = bal - :new.amount

where maccno = :new.taccno;

elsif :new.type = 'deposit' then

update master

set balance = bal + :new.amount

where maccno = :new.taccno;

end if;

else

insert into master (maccno, name, balance)

values (:new.taccno, :new.tname, :new.amount);

end if;

end if;

insert into audit (accno, type, user\_id, sys\_date, amount)

values (:new.taccno, :new.type, user, sysdate, :new.amount);

end;

/

1. For the following tables and view in the database:

**STUDENT (SEATNO, NAME, MARKS1, MARKS2, MARKS3)**

**RESULT (SEATNO, AVERAGE)**

**VTEMP (SEATNO, VDATE, ACTION)**

**V1 (SEATNO, NAME, AVERAGE)**

For every DML against the view V1, write a instead-of trigger that will insert the details regarding the SEATNO, the date on which the view is being modified as well as the action (Insert / Update / Delete) in the table VTEMP.

create or replace trigger v1\_trigger

instead of insert or update or delete on v1

for each row

begin

if inserting then

insert into vtemp (seatno, vdate, action)

values (:new.seatno, sysdate, 'insert');

elsif updating then

insert into vtemp (seatno, vdate, action)

values (:old.seatno, sysdate, 'update');

elsif deleting then

insert into vtemp (seatno, vdate, action)

values (:old.seatno, sysdate, 'delete');

end if;

end;

/

1. Write system triggers that stores details about the timings of login and logout on your account as well as details about every DDL that is executed once you are logged in. Create the necessary audit tables to store the required information.

create table login\_audit (

login\_time TIMESTAMP,

logout\_time TIMESTAMP

);

create table ddl\_audit (

ddl\_statement VARCHAR2(4000),

ddl\_time TIMESTAMP

);

create or replace trigger login\_trigger

after logon on database

begin

insert into login\_audit (login\_time) values (SYSTIMESTAMP);

end;

/

create or replace trigger logout\_trigger

before logoff on database

begin

update login\_audit

set logout\_time = SYSTIMESTAMP

where logout\_time is null;

end;

/

create or replace trigger ddl\_trigger

after ddl on database

declare

begin

insert into ddl\_audit (ddl\_statement, ddl\_time)

values (ora\_sql\_txt(SQL\_TEXT), SYSTIMESTAMP);

end;

/