S.Y. Diploma : Sem. III

[CO/CM/CW]

Database Management System [DBMS]

Time: 3 Hrs.] Prelim Question Paper Solution **[Marks : 70**

Q.1 Attempt any FIVE of the following : [10]

Q.1(a) List the transaction states. [2]

Ans.: Following are the five transaction states

(i) Active
(ii) Partially Committed
(iii) Committed
(iv) Failed
(v) Aborted

Q.1(b) State any four database application. [2]

Ans.: Following are four database application

(1) Banking application
(2) University
(3) University
(4) Online Retailers
(5) Hospital

Q.1(c) Define weak entity set. [2]

Ans.: (i) An entity set is said to be a weak entity set when it lacks sufficient attributes to form a primary key. It depends or is owned by another entity set called as owner entity set.

(ii) It is existence dependent on owner entity set.

Q.1(d) Define normalization. Enlist its type. [2]

Ans.: Normalization

Normalization is a process of analyzing given relational schema based on its functional dependencies and primary keys/candidate keys to achieve the following desired properties.

Following are the two objectives of normalization:

(i) Minimize data redundancy.
(ii) Minimize insert, update, delete anomalies

Types :

1 NF,
2 NF
3 NF, BCNF, 4NF, 5NF

Q.1(e) Define the following **[2]**

(i) Candidate Key **(ii) Super Key**

Ans.: (i) Candidate key: The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For Example, STUD_NO in STUDENT relation.

(ii) Super Key: The set of attributes which can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME) etc.

Q.1(f) Enlist four aggregate functions. **[2]**

Ans.: Aggregate Functions

Aggregate functions are the functions that take a collection of values as input and return a single value.

Avg function : The avg function computes the column's average value.

Min & Max functions : The min and max functions return the minimum and maximum values for the specified columns

e.g.: SQL > select max (salary), min (salary) from employee;

Sum function : The sum function computes the column's total value.

SQL> **Select sum** (salary) **from** employee;

Count function : The count function counts the number of rows. They are of two forms

Count (*): It counts all the rows in a table that satisfy any specified criteria.

Count (column name): It counts all rows in a table that have a non-null value for the column name and satisfy the specified criteria.

e.g. : **Select count (*) from** employee.

Q.1(g) Draw PL/SQL block structure. **[2]**

Ans.:

```
DECLARE --optional
    <declarations>

BEGIN  --mandatory
    <executable statements. At least one executable statement is mandatory>

EXCEPTION --optional
    <exception handles>

END;  --mandatory
/
```

Q.2 Attempt any THREE of the following : [12]

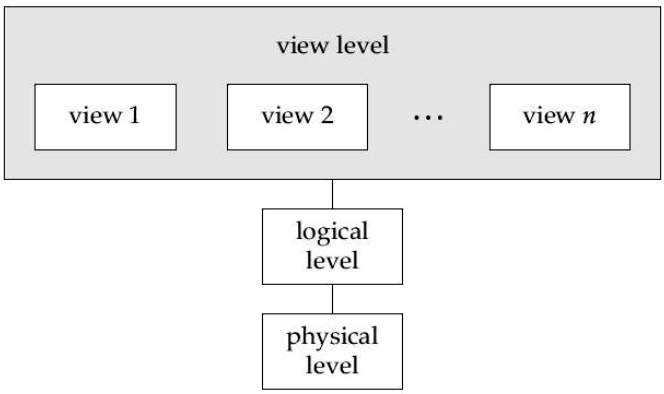
Q.2(a) Explain the levels of Data Abstraction. [4]

Ans.: A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

Data Abstraction : Developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:
Physical level : The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

Logical level : The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures
This is referred to as physical data independence. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

View level : The highest level of abstraction describes only part of the entire database.
The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

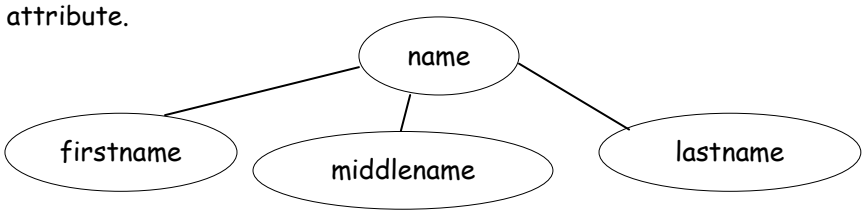


Q.2(b) Explain different types of attributes. [4]

Ans.: An attribute can be characterized by the following attribute types:

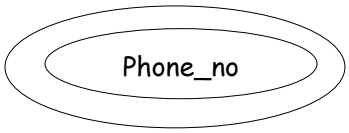
(1) Simple attributes: An attribute which cannot be divided into further sub parts is called as Simple attribute. For example roll no, id attribute of an student entity cannot be divided into further sub parts.

(2) Composite attribute: An attribute which can be divided into further sub parts is called as Composite attribute. For example name attribute of student can be divided into firstname, middlename and lastname attribute.

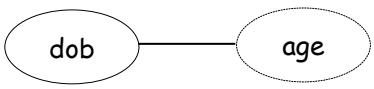


(3) Single valued attribute: An attribute holding only one value is called as single valued attribute. For example, roll no.,id

(4) Multivalued attribute: An attribute having multiple values is termed as multivalued attribute. For example phone no attribute can have two values one being residential number and another mobile number. It is represented by double lined oval as follows



(5) Derived attribute: An attribute which can be derived from another attribute is called as derived attribute. For example age attribute can be derived from date of birth attribute. It is represented by dotted oval.



(6) Null attribute: An attribute holding no value i.e null is called as null attribute.

Q.2(c) Explain the different types of functional dependencies. [4]

Ans.: Functional dependencies

(i) A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the determinant and the other attribute is called the determined. For each value of the determinant there is associated one and only one value of the determined.

(ii) If A is the determinant and B is the determined then we say that A functionally determines B and graphically represent this as A → B. The symbols A à B· can also be expressed as B is functionally determined by A.

(iii) Following are various kinds of functional dependencies:

- **Fully Functional Dependence (FFD)**
  Fully Functional Dependence (FFD) is defined, as Attribute Y is FFD on attribute" X, if it is functionally dependent on X and not functionally dependent on any proper subset of X.
- **Partial functional dependency**
  A **partial functional dependency** is a functional dependency where the determinant is a part of primary/candidate key determining other non prime attributes.
  Eg: For given relation R(A,B,C) if AB is a candidate key then B→C is a partial dependency because B is a part of candidate key AB
- **Transitive functional dependency**
  A **transitive functional dependency** is a functional dependency where the determinant consists of non-key attributes and the determined also consists of non-key attributes.
  A→B [B depends on A] & B → C [C depends on B]
  Then A → C[C depends on A] can be derived.

Q.2(d) Distinguish any four points between PROCEDURE and FUNCTION. [4]

Ans.:

Sr. No.	Procedure	Function
1	Procedure does not return a value	Function always returns a value
2	Procedures cannot be called from functions	Functions can be called from procedures
3	Exception can be handled by try-catch block in a Procedure	Try-catch block cannot be used in a Function.

4	Procedure allows SELECT as well as DML(INSERT/UPDATE/DELETE) statement	Function allows only SELECT statement in it.
5	Stored Procedures cannot be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section	Function can be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section
6	Syntax: CREATE [OR REPLACE] PROCEDURE procedure_name [(parameter [,parameter])] IS [declaration_section] BEGIN executable_section [EXCEPTION exception_section] END [procedure_name];	Syntax: CREATE [OR REPLACE] FUNCTION function_name [(parameter_name [IN | OUT | IN OUT] type [, ...])] RETURN return_datatype {IS | AS} BEGIN < function_body > END [function_name];

Q.3 Attempt any THREE of the following : [12]

Q.3(a) Explain GROUP BY and ORDER BY CLAUSE with syntax and example. [4]

Ans.: GROUP BY clause:

It is used to arrange identical data into groups.

It is often used with aggregate functions to group the result set by one or more columns.

Syntax:

SELECT column_names FROM table_name WHERE condition GROUP BY column_names;

Example: To display the number of customers in each country.

Assume table

CUSTOMERS(customer_id,cust_name,phone,email,city,country,pincode)

SELECT COUNT(customer_id),country FROM customers GROUP BY country;

ORDER BY clause:
This clause is used to sort the result set in ascending or descending order
It sorts the records in ascending order by default
Syntax:
SELECT column_name1, column_name2,… FROM table_name ORDER BY column_name1 asc|desc;
Example:
SELECT * FROM customers ORDER BY country desc;

Q.3(b) Write and explain the syntax for creating PROCEDURES. [4]

Ans.: A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows :

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
   < procedure_body >
END procedure_name;
```

Where,
procedure-name specifies the name of the procedure.
[OR REPLACE] option allows the modification of an existing procedure.

The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.

procedure-body contains the executable part.
The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example: Creating Procedure and calling it using EXEC

```
CREATE OR REPLACE PROCEDURE welcome_msg (p_name IN VARCHAR2)
IS
BEGIN
dbms_output.put_line ('Welcome '|| p_name);
END;
/
EXEC welcome_msg ('world');
```

Q.3(c) Explain user defined exception handling with the help of example. [4]

Ans.: User Defined Exception

In user defined exception, the programmer can create their own exception and handle them. They can be created at a subprogram level in the declaration part. These exceptions are visible only in that subprogram.

Example [1 mark]

- Divide non-negative integer x by y such that the result is greater than or equal to 1. From the given question we can conclude that there exist two exceptions.
  - Division be zero.
  - If result is greater than or equal to 1 means y is less than or equal to x.

```
DECLARE
    x int:=&x; /*taking value at run time*/
    y int:=&y;
    div_r float;
    exp1 EXCEPTION;
    exp2 EXCEPTION;
BEGIN
    IF y=0 then
        raise exp1;
ELSEIF y > x then
        raise exp2;
ELSE
    div_r:= x / y;
    dbms_output.put_line('the result is '||div_r);

END IF;

EXCEPTION
    WHEN exp1 THEN
        dbms_output.put_line('Error');
        dbms_output.put_line('division by zero not allowed');

    WHEN exp2 THEN
        dbms_output.put_line('Error');
        dbms_output.put_line('y is greater than x please check the
        input');

    END;
```

Q.3(d) Explain ACID properties of transaction. [4]

Ans.: **(i) Atomicity :** Atomicity is based on the concept that each transaction be "all or nothing": if any one part of the transaction in a sequence fails, then the entire transaction fails, and there will be no change in database state.

This property states that, either all operations contained by a transaction are done successfully or none of them complete at all. To maintain the consistency of data this property is very useful.

(ii) Consistency : The consistency property ensures that the transaction executed on the database system will bring the database from original consistent state to another.

(iii) Isolation : The isolation property ensures that the system state should be same that would be obtained if transactions were executed sequentially, i.e., one after the other. The effect of any incomplete transaction should not be visible to other transaction. This can be achieved isolation.

(iv) Durability : The durability property ensures that after transaction is committed successfully the updates made should remain permanent in the database even in the event of power loss, crashes or errors.

Q.4 Attempt any THREE of the following : [12]

Q.4(a) List and explain TCL commands. [4]

Ans.: **Transaction Control Language (TCL)** commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

COMMIT command

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

COMMIT;

ROLLBACK command

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a save point in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

ROLLBACK TO savepoint_name;

SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,
SAVEPOINT savepoint_name;

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Q.4(b) Difference between system privileges and object privileges. [4]

Ans.:

Sr. No.	System Privileges	Table Level Privileges (Object Level)
(i)	System Privileges are normally granted by a DBA to users.	Object privileges means privileges on objects such as tables, views, synonyms, procedure. These are granted by owner of the object.
(ii)	This privileges allows a user to manage database and server.	This privileges allows a user to perform certain action upon certain database objects
(iii)	**List of privileges** • CREATE USER • CREATE TABLE • CREATE SESSION	**List of privileges** • SELECT •INSERT,UPDATE,DELETE • EXECUTE

(iv)	To get information about system level privileges SELECT * FROM USER_SYS_PRIVS SELECT * FROM ROLE_SYS_PRIVS	To get information about object level privileges SELECT OWNER,TABLE_NAME,PRIVILEGE FROM USER_TAB_PRIVS SELECT * FROM ROLE_TAB_PRIVS
(v)	**Syntax** GRANT privileges TO username;	**Syntax** GRANT privileges ON object TO username;
(vi)	**Example** GRANT CREATE SESSION, CREATE PROCEDURE TO PARAG;	**Example** GRANT SELECT, INSERT ON EMPLOYEE TO PARAG;

Q.4(c) Explain the explicit cursor with the help of an example. [4]

Ans.: Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is
CURSOR cursor_name IS select_statement;

Working with an explicit cursor includes the following steps:

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example:

```
CURSOR c_customers IS
    SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows :

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows :

FETCH c_customers INTO c_id, c_name, c_addr;

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows :

CLOSE c_customers;

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
    c_id customers.id%type;
    c_name customerS.No.ame%type;
    c_addr customers.address%type;
    CURSOR c_customers is
        SELECT id, name, address FROM customers;
BEGIN
    OPEN c_customers;
    LOOP
    FETCH c_customers into c_id, c_name, c_addr;
        EXIT WHEN c_customers%notfound;
        dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
    END LOOP;
    CLOSE c_customers;
END;
/
```

Q.4(d) Explain 2NF with example. **[4]**

Ans.: 2NF

A database is in second normal form if it satisfies the following conditions:

- It is in first normal form.
- All non-key attributes are fully functional dependent on the primary key.

In a table, if attribute B is functionally dependent on A, but is not functionally dependent on a proper subset of A, then B is considered fully functional dependent on A. Hence, in a 2NF table, all non-key attributes cannot be dependent on a subset of the primary key. Note that if the primary key is not a composite key, all non-key attributes are always fully

functional dependent on the primary key. A table that is in 1st normal form and contains only a single key as the primary key is automatically in 2nd normal form.

2nd Normal Form Example

Consider the following example:

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

To bring this table to second normal form, we break the table into two tables, and now we have the following:

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

Now, in the table [TABLE_STORE], the column [Purchase Location] is fully dependent on the primary key of that table, which is [Store ID].

Q.4(e) List and Explain any five EF Codd's rules. [4]

Ans.: 1) Foundation Rule

A relational database management system must manage its stored data using only its relational capabilities.

2) Information Rule

All information in the database should be represented in one and only one way - as values in a table.

3) Guaranteed Access Rule

Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

4) Systematic Treatment of Null Values

Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.

5) Dynamic On-line Catalog Based on the Relational Model

The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.

6) Comprehensive Data Sublanguage Rule

A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible:

(a) data definition
(b) view definition
(c) data manipulation (interactive and by program)
(d) integrity constraints
(e) authorization
(f) transaction boundaries (begin, commit, and rollback).

7) View Updating Rule

All views that are theoretically updateable are also updateable by the system.

8) High-level Insert, Update, and Delete

The capability of handling a base relation or a derived relation as a single operand applies nor only to the retrieval of data but also to the insertion, update, and deletion of data.

9) Physical Data Independence

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

10) Logical Data Independence

Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

11) Integrity Independence

Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

12) Distribution Independence

The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.

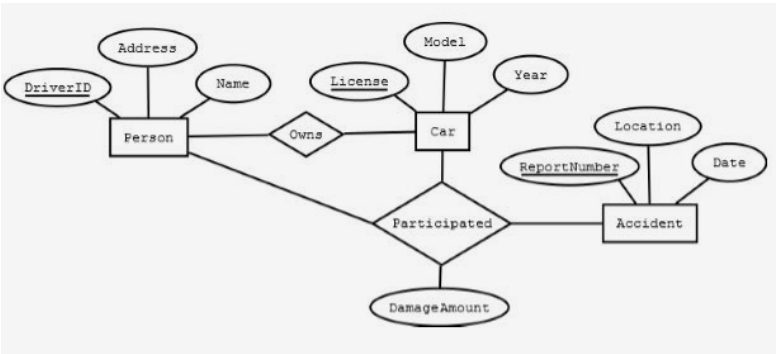
13) Nonsubversion Rule

If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

Q.5 Attempt any TWO of the following : [12]

Q.5(a) Draw ER diagram for a car insurance company. [6]

Ans.:



Q.5(b) Normalize database Student_Grade_Report (StudentNo, StudentName, Major, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade) upto 3NF [6]

Ans.: **Student_Grade_Report** (StudentNo, StudentName, Major, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

- In the Student Grade Report table, the repeating group is the course information. A student can take many courses.

- Remove the repeating group. In this case, it's the course information for each student.
- Identify the PK for your new table.
- The PK must uniquely identify the attribute value (StudentNo and CourseNo).
- After removing all the attributes related to the course and student, you are left with the student course table (**StudentCourse**).
- The Student table (**Student**) is now in first normal form with the repeating group removed.
- The two new tables are shown below.

Student (StudentNo, StudentName, Major)
StudentCourse (StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

How to update 1NF anomalies
StudentCourse (StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

- To add a new course, we need a student.
- When course information needs to be updated, we may have inconsistencies.
- To delete a *student,* we might also delete critical information about a course.

Second Normal Form (2NF)
For the *second normal form*, the relation must first be in 1NF. The relation is automatically in 2NF if, and only if, the PK comprises a single attribute.

If the relation has a composite PK, then each non-key attribute must be fully dependent on the entire PK and not on a subset of the PK (i.e., there must be no partial dependency or augmentation).

Process for 2NF
To move to 2NF, a table must first be in 1NF.

- The Student table is already in 2NF because it has a single-column PK.
- When examining the Student Course table, we see that not all the attributes are fully dependent on the PK; specifically, all course information. The only attribute that is fully dependent is grade.
- Identify the new table that contains the course information.
- Identify the PK for the new table.
- The three new tables are shown below.

Student (StudentNo, StudentName, Major)
CourseGrade (StudentNo, CourseNo, Grade)
CourseInstructor (CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation)

How to update 2NF anomalies

- When adding a new instructor, we need a course.
- Updating course information could lead to inconsistencies for instructor information.
- Deleting a course may also delete instructor information.

Third Normal Form (3NF)

To be in *third normal form*, the relation must be in second normal form. Also all transitive dependencies must be removed; a non-key attribute may not be functionally dependent on another non-key attribute.

Process for 3NF

- Eliminate all dependent attributes in transitive relationship(s) from each of the tables that have a transitive relationship.
- Create new table(s) with removed dependency.
- Check new table(s) as well as table(s) modified to make sure that each table has a determinant and that no table contains inappropriate dependencies.
- See the four new tables below.

Student (StudentNo, StudentName, Major)
CourseGrade (StudentNo, CourseNo, Grade)
Course (CourseNo, CourseName, InstructorNo)
Instructor (InstructorNo, InstructorName, InstructorLocation)

Q.5(c) Write SQL query for following consider table [6]
EMP(empno, deptno, ename, salary, Designation, joiningdate, DOB,city)
(i) Display names of employees whose experience is more than 10 years
(ii) Display age of employees
(iii) Display average salary of all employee
(iv) Display name of employee who earned highest salary

Ans.: (i) Display names of employees whose experience is more than 10 years

```
Select ename from EMP where to_char(sysdate,'YYYY')-
to_char(joiningdate,'YYYY')>10;
```

OR

Select ename from EMP where DATEDIFF(YEAR, joiningdate, GETDATE())>10

(ii) Display age of employees

Select emp_no,ename, to_char(sysdate,'YYYY')-to_char(dob,'YYYY')as Age from EMP

OR

Select emp_no,emp_name, DATEDIFF(YEAR, DOB, GETDATE()) as Age from EMP

(iii) Display average salary of all employee

Select avg(salary) as Average_salary from EMP

(iv) Display name of employee who earned highest salary

Select emp_name,salary from EMP where salary =(select max(salary) from EMP).

Q.6 Attempt any TWO of the following : [12]

Q.6(a) Create table [6]

EMP(empno, deptno, ename, salary, Designation, joiningdate, DOB, city).

(i) Insert one row into the table
(ii) Save the data
(iii) Insert second row into the table
(iv) Undo the insertion of second row
(v) Insert two rows into the table
(vi) Create Savepoint s1
(vii) Insert one row into the table
(viii) Undo upto savepoint s1

Ans.: CREATE TABLE EMP
(EMPNO INT, DEPTNO,ENAME VARCHAR(20),SALARY FLOAT,DESIGNATION VARCHAR(100),JOININGDATE DATE,DOB DATE,CITY VARCHAR(20));

(i) Insert one row into the table

INSERT INTO EMP VALUES(1,10,'ANIL',10000,'HR','01-02-2006','12-03-1990','MUMBAI');

(ii) Save the data

COMMIT;

(iii) Insert second row into the table

```
INSERT INTO EMP VALUES (2,10,'SUJA',12000,'SALES
REPRESENTATIVE','01-02-2009','12-03-1987','MUMBAI');
```

(iv) Undo the insertion of second row

```
ROLLBACK;
```

(v) Insert two rows into the table

```
INSERT INTO EMP VALUES (2,10,'SUJA',12000,'SALES
REPRESENTATIVE'','11-08-2009','21-03-1987','MUMBAI');

INSERT INTO EMP VALUES(3,20,'LEENA',15000,'PR MANAGER',
'01-02-2010','12-05-1974','MUMBAI');
```

(vi) Create Savepoint s1

```
SAVEPOINT S1;
```

(vii) Insert one row into the table

```
INSERT INTO EMP VALUES(4,20,'AJAY',18000,'CLERK','25-06-
2010','18-08-1974','MUMBAI');
```

(viii) Undo upto savepoint s1

```
ROLLBACK TO S1
```

Q.6(b) Write a PL/SQL program to check whether specified employee is present in EMP table or not. Accept empno from user. If employee does not exist display message using exception handling. [6]

Ans.:

```
Accept x number prompt 'Please enter emp no:
DECLARE
Emp_no number;
BEGIN

Emp_no:=&x;
Select * from EMP where emp_no=Emp_no;

EXCEPTION
    WHEN no_data_found THEN
        dbms_output.put_line('No such employee!');

END;
/
```

Q.6(c) Write a PL/SQL program to print Fibonacci series. **[6]**

Ans.:

```
declare
    first number:=0;
    second number:=1;
    third number;
    n number:=&n;
    i number;
begin
    dbms_output.put_line('Fibonacci series is:');
    dbms_output.put_line(first);
    dbms_output.put_line(second);

    for i in 2..n
    loop
        third:=first + second;
        first:=second;
        second:=third;
        dbms_output.put_line(third);
    end loop;
end;
/
```

❑ ❑ ❑ ❑ ❑