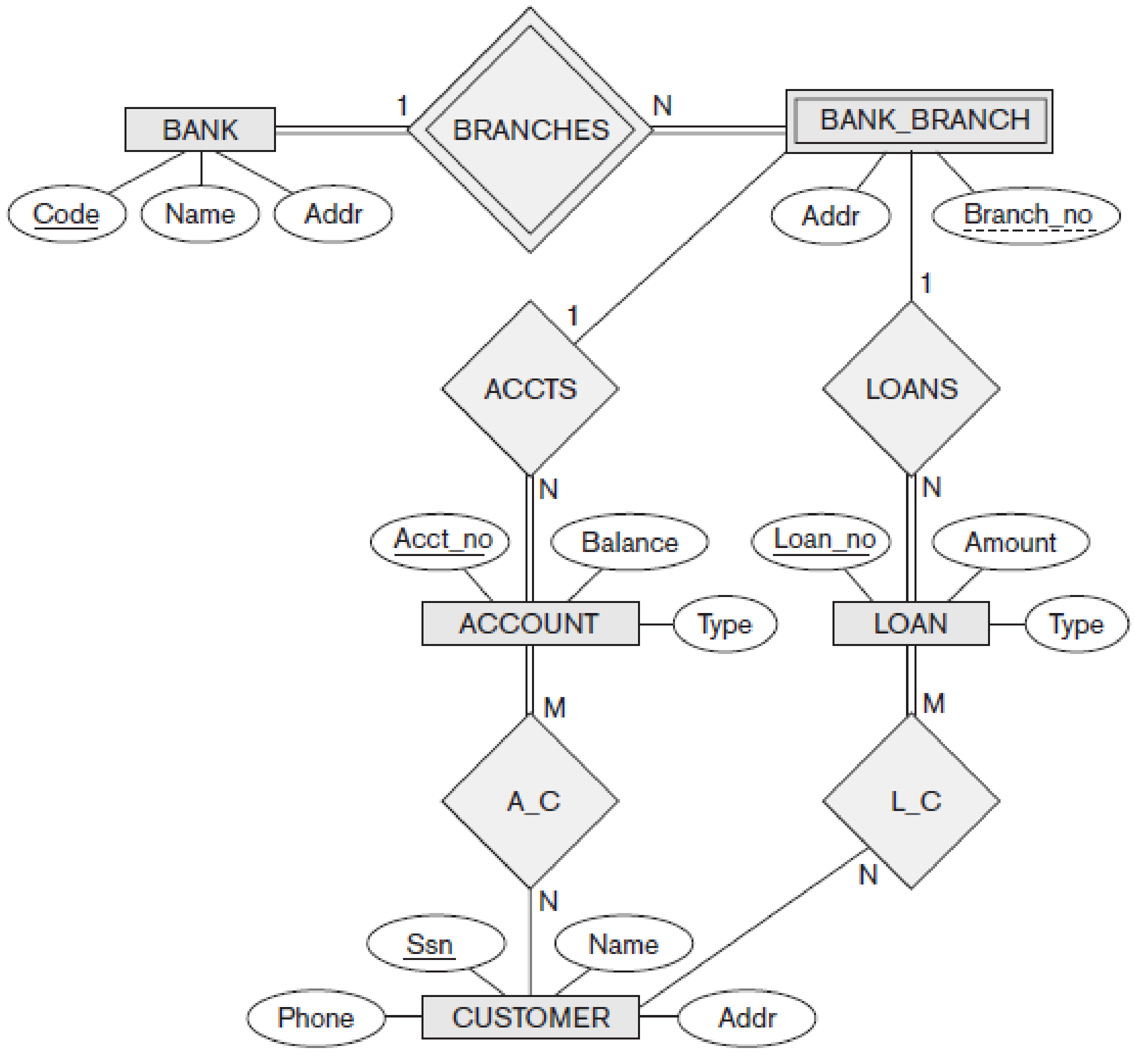
Sub:	Database Management Systems					Sub Code:	17CS53	
Date:	14/10/19	Duration:	90 mins	Max Marks:	50	Sem /Sec:	V A,B,C	OBE

		CO	RBT
	Answer any FIVE FULL QUESTIONS		
1	**Draw an ER diagram for BANK database schema with at least five entities also specify primary key and structural constraints(Relationship and participation constraint).**	CO1	L3

Answer: Diagram

An ER diagram for a BANK database schema.



2 A

Explain the informal design guidelines used as measures to determine the quality of relation schema design.

CO4 L2

Answer:

Four *informal guidelines* that may be used as *measures to determine the quality* of relation schema design:
a.Making sure that the semantics of the attributes is clear in the schema
b.Reducing the redundant information in tuples
c. Reducing the NULL values in tuples
d.Disallowing the possibility of generating spurious tuples.

(a)Imparting Clear Semantics to Attributes in Relations
The semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple.While designing a schema, assign simple but meaningful name for the schema and attributes so that schema can be easily explained.Semantics of attributes should be easy to interpret.Name should convey maximum meaning as possible.

Guideline 1. Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, it is straightforward to explain its meaning.
(b)Redundant Information in Tuples and Update Anomalies
When information is stored redundantly,
-Wastes storage space
-Causes problems with update anomalies(insertion ,deletion and updation anomalies
In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for *every employee who works for that department.* Storing natural joins of base relations leads to an additional problem referred to as update anomalies. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.
Insertion Anomalies. Insertion anomalies can be differentiated into two types,

a.To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs (if the employee does not work for a department as yet). For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are *consistent* with the corresponding values for department 5 in other tuples in EMP_DEPT

b. It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee. This violates the entity integrity for EMP_DEPT because its primary key Ssn cannot be null.

Deletion Anomalies
If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost inadvertently from the database.
Modification Anomalies
In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of *all* employees who work in that department.

Guideline 2. Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present,4 note them clearly and make sure that the programs that update the database will operate correctly.
(c)NULL Values in Tuples
If many of the attributes do not apply to all tuples in the relation, we end upwith many NULLs in those tuples.NULLs can have multiple interpretations,such as the following:

- The attribute *does not apply* to this tuple.
- The attribute value for this tuple is *unknown*.
- The value is *known but absent*; that is, it has not been recorded yet.

Guideline 3. As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

(d)Generation of Spurious Tuples
Guideline 4. Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

2 b **Explain how Group by clause works. What is the difference between "where" and "having"?** CO3 L2

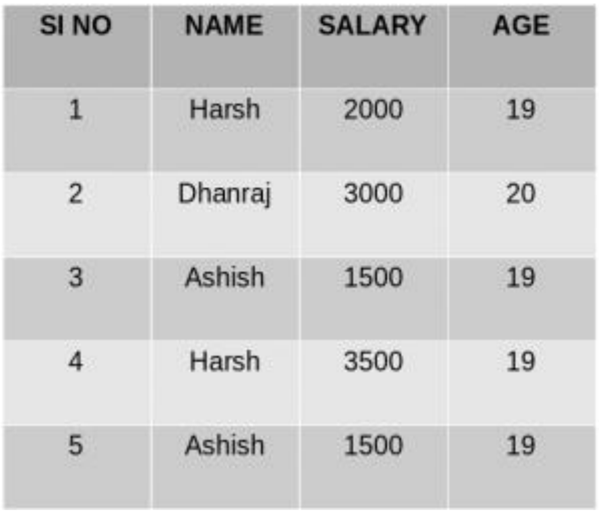
The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.

Syntax:

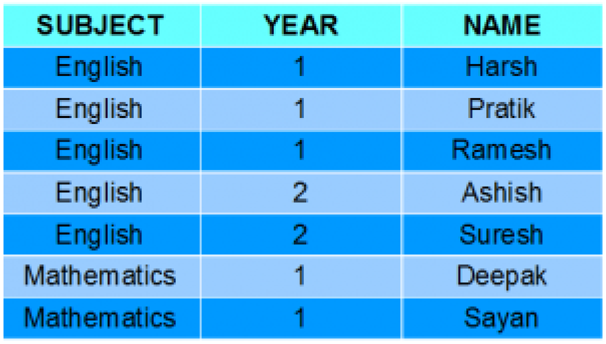
```
SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
ORDER BY column1, column2;
```

Employee



SI NO	NAME	SALARY	AGE
1	Harsh	2000	19
2	Dhanraj	3000	20
3	Ashish	1500	19
4	Harsh	3500	19
5	Ashish	1500	19

Student

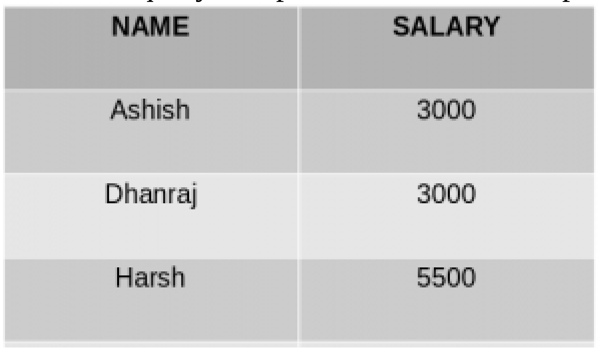


SUBJECT	YEAR	NAME
English	1	Harsh
English	1	Pratik
English	1	Ramesh
English	2	Ashish
English	2	Suresh
Mathematics	1	Deepak
Mathematics	1	Sayan

Example:

- **Group By single column**: Group By single column means, to place all the rows with same value of only that particular column in one group. Consider the query as shown below:
- SELECT NAME, SUM(SALARY) FROM Employee
- GROUP BY NAME;

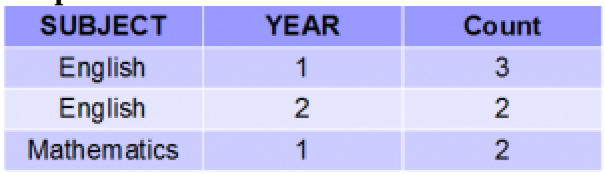
  The above query will produce the below output:



NAME	SALARY
Ashish	3000
Dhanraj	3000
Harsh	5500

- As you can see in the above output, the rows with duplicate NAMEs are grouped under same NAME and their corresponding SALARY is the sum of the SALARY of duplicate rows. The SUM() function of SQL is used here to calculate the sum.
- **Group By multiple columns**: Group by multiple column is say for example, **GROUP BY column1, column2**. This means to place all the rows with same values of both the columns **column1** and **column2** in one group. Consider the below query:
- SELECT SUBJECT, YEAR, Count(*)
- FROM Student
- GROUP BY SUBJECT, YEAR;

  Output:



SUBJECT	YEAR	Count
English	1	3
English	2	2
Mathematics	1	2

- As you can see in the above output the students with both same SUBJECT and YEAR are placed in same group. And those whose only SUBJECT is same but not YEAR belongs to different groups. So here we have grouped the table according to two columns or more than one column.

HAVING Clause

We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groups?

This is where HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

Syntax:

```
SELECT column1, function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

HAVING condition

ORDER BY column1, column2;
```

function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.

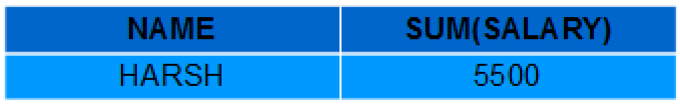
Example:

```
SELECT NAME, SUM(SALARY) FROM Employee

GROUP BY NAME

HAVING SUM(SALARY)>3000;
```

Output:



NAME	SUM(SALARY)
HARSH	5500

As you can see in the above output only one group out of the three groups appears in the result-set as it is the only group where sum of SALARY is greater than 3000. So we have used HAVING clause here to place this condition as the condition is required to be placed on groups not columns.

3 a

Discuss how each of the following constructs used in SQL - (i)views and their updatability

CO3 L2

In **SQL**, a view is a virtual table based on the result-set of an **SQL** statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

Example

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = "Brazil";
```

We can query the view above as follows:

Example

```
SELECT * FROM [Brazil Customers];
```

Try it Yourself »

Update View

The SQL UPDATE VIEW command can be used to modify the data of a view.

All views are not updatable. So, UPDATE command is not applicable to all views. An updatable view is one which allows performing a UPDATE command on itself without affecting any other table.

A view can be updated with the CREATE OR REPLACE VIEW command.

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The following SQL adds the "City" column to the "Brazil Customers" view:

Example

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = "Brazil";
```

3 b	**Discuss how each of the following constructs used in SQL - (ii) Triggers**	CO3	L2

Answer:

Triggers are stored procedures for the actions to be performed on certain condition. It is defined in such a way that it will run automatically, before or after some events such as INSERT,UPDATE OR DELETE.

Syntax:

```
Create trigger <trigger_name>
[before/after]
{insert/update/delete}
On [table_name]
[for each row]
[trigger-body]
```

Example:

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
ON EMPLOYEE
```

```
FOR EACH ROW
WHEN ( NEW.SALARY> ( SELECT SALARY FROM EMPLOYEE
WHERE SSN = NEW.SUPERVISOR_SSN ) )
INFORM_SUPERVISOR(NEW.Supervisor_ssn,
NEW.Ssn );
```

A typical trigger which is regarded as an ECA (Event,Condition, Action) rule has three components:

1. The event(s): These are usually database update operations that are explicitly applied to the database. In this example the events are: inserting a new employee record, changing an employee's salary, or changing an employee's supervisor . These events are specified after the keyword BEFORE in this example, which means that the triggers should be executed before the triggering operation is executed. An alternative is to use the keyword AFTER, which specifies that the trigger should be executed after the operation specified in the event is completed.
2. The condition that determines whether the rule action should be executed: Once the triggering event has occurred, an *optional* condition may be evaluated.If*no condition* is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only *if it evaluatesto true* will the rule action be executed. The condition is specified in the WHEN clause of the trigger.
3. The action to be taken: The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed.

Triggers can be used in various applications, such as maintaining database consistency, monitoring database updates, and updating derived data automatically.

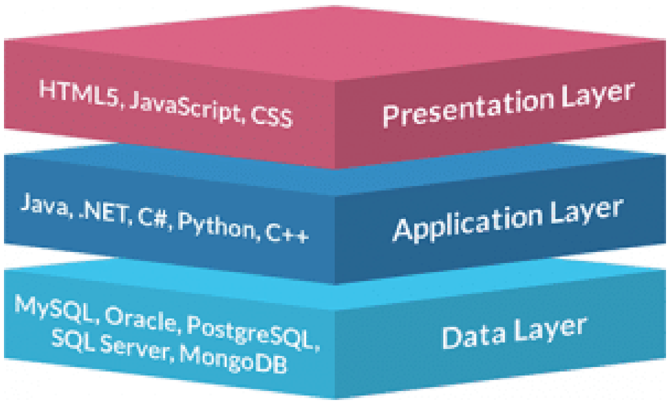
4 a

Draw and explain 3 tier-architecture

CO3 L2

A 3-tier architecture is a type of software architecture which is composed of three "tiers" or "layers" of logical computing. They are often used in applications as a specific type of client-server system. 3-tier architectures provide many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers. Doing so gives greater flexibility to development teams by allowing them to update a specific part of an application independently of the other parts. This added flexibility can improve overall time-to-market and decrease development cycle times by giving development teams the ability to replace or upgrade independent tiers without affecting the other parts of the system.

What Do the 3 Tiers Mean?


HTML5, JavaScript, CSS
Presentation Layer
Java, .NET, C#, Python, C++
Application Layer
MySQL, Oracle, PostgreSQL, SQL Server, MongoDB
Data Layer

- **Presentation Tier-** The presentation tier is the front end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user. This tier is often built on web technologies such as HTML5, JavaScript, CSS, or through other popular web development frameworks, and communicates with others layers through API calls.
- **Application Tier-** The application tier contains the functional business logic which drives an application's core capabilities. It's often written in Java, .NET, C#, Python, C++, etc.
- **Data Tier-** The data tier comprises of the database/data storage system and data access layer. Examples of such systems are MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

4 b

CO4 L2

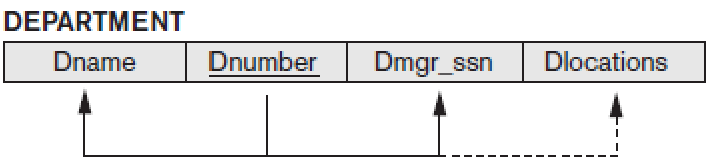
Explain 1NF,2NF.

Answer:

First Normal Form

First normal form (1NF)is now considered to be part of the formal definition of a relation in the basic (flat) relational model. it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only *atomic* (simple, indivisible) *values* and that the value of any attribute in a tuple must be a *single value* from the domain of that attribute. 1NF disallows *relations within relations* or *relations as attribute values within tuples*. The only attribute values permitted by 1NF are single atomic (or indivisible) values.

(a)



(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

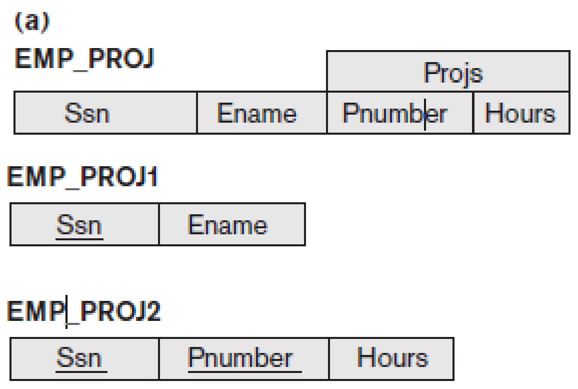
DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

There are three main techniques to achieve first normal form for such a relation:

1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this newly formed relation is the combination {Dnumber, Dlocation}, as shown in Figure 14.2. A distinct tuple in DEPT_LOCATIONS exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT, as shown in Figure .In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing *redundancy* in the relation and hence is rarely adopted.
3. If a *maximum number of values* is known for the attribute—for example, if it is known that *at most three locations* can exist for a department—replace the Dlocations attribute by three

atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing *NULL values* if most departments have fewer than three locations.
The schema of this EMP_PROJ relation can be represented as follows: EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})



The existence of more than one multivalued attribute in one relation must be handled carefully. As an example, consider the following non-1NF relation:
PERSON (Ss#, {Car_lic#}, {Phone#})

The right way to deal with the two multivalued attributes in PERSON is to decompose it into two separate relations.

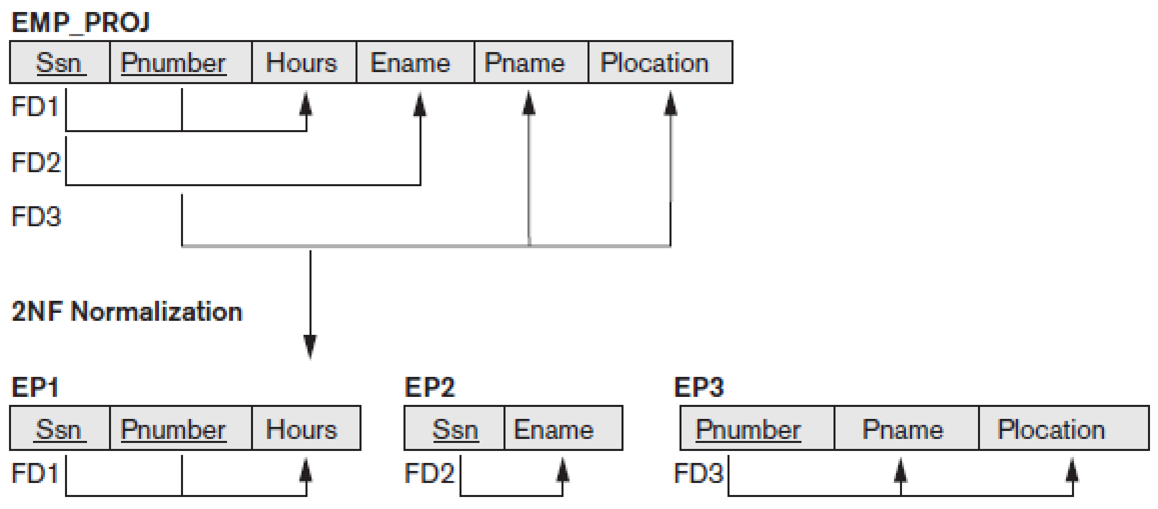
P1(Ss#, Car_lic#) and P2(Ss#, Phone#).

2NF (Second Normal Form)

Second normal form (2NF) is based on the concept of *full functional dependency*. A functional dependency *X* → *Y* is a full functional dependency if removal of anyattribute *A* from *X* means that the dependency does not hold anymore.

Definition. A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. If a relation schema is not in 2NF, it can be second normalized or 2NF normalized into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent.



5 a

Write relational algebra expressions for the following problems:

CO2 L3

Ans-5(a) Relational Algebra Queries:

(i)Find the names of employees who work on all projects controlled by department no.4

DEPT5 PROJS(PNO) ← πP NUMBER(σDNUM=4(PROJECT))
EMP PROJ(SSN, PNO) ← πESSN,P NO(WORKS ON)
RESULT EMP SSNS ← EMP PROJ ÷ DEPT PROJS
RESULT ←πLNAME,F NAME(RESULT EMP SSNS ∗ EMPLOY EE)

(ii) Retrieve the name and address of all employees who work for the 'Research' department
RESEARCH DEPT ← σDNAME='Research'(DEPARTMENT)
RESEARCH EMPS ← (RESEARCH_ DEPT ⋈DNUMBER=DNO (EMPLOY EE))
RESULT ←πF NAME,LNAME,ADDRESS(RESEARCH_EMPS)

(iii) List the name of all managers with at least one dependent.
MGRS(SSN) ← πMGRSSN (DEPARTMENT)
EMPS _WITH _DEPS(SSN) ←πESSN (DEPENDENT)
MGRS_ WITH _DEPS ← (MGRS ∩ EMPS_WITH_DEPS)
RESULT ← πLNAME,F NAME(MGRS_WITH _DEPS ∗ EMPLOYEE)

5 b

Write SQL queries for the following problems:

CO3 L3

5(b) SQL queries:

(i) List the name of all employees who have two or more dependents.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
 WHERE (SELECT COUNT (*) FROM DEPENDENT WHERE SSN = ESSN) >= 2;
```

(ii) Retrieve the names of employees who have no dependents(USE EXISTS)

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE NOT EXISTS (SELECT * FROM DEPENDENT WHERE SSN = ESSN);
```

(iii) Print details of department that has less than 5 people working in it

```
Select D.*,count(E.ssn)
From DEPARTMENT D,EMPLOYEE E
 where E.dno=D.dnumber
 Group by D.dnumber
 Having count(E.ssn)<5;
```

6 a

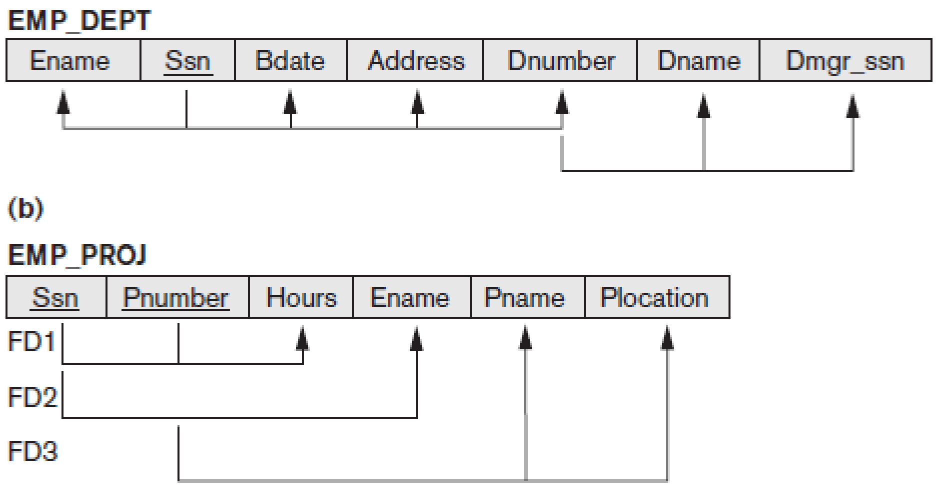
Explain functional dependency with example and demonstrate its representation.

CO4 L2

A functional dependency is a constraint between two sets of attributes from the Database.
Definition. A functional dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y].

The values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X componentof a tuple uniquely (or functionally) determine the values of the Y component. The abbreviation for functional dependency is FD or f.d.The set ofattributes X is called the left-hand side of the FD, and Y is called the right-hand side. A functional dependency is a property of the semantics or meaning of the attributes.

Diagrammatic notation for displaying FDs: Each FD is displayed as a horizontal line. The left-hand-side attributes of the FD are connected by vertical lines to the line representing the FD, whereas the right-hand-side attributes are connected by the lines with arrows pointing toward the attributes.



6 b

<span style="color:red">**What are the different kinds of join operations in relational algebra? Explain each with example**</span> CO2 L1

Join Operations

Join operation is essentially a cartesian product followed by a selection criterion.

Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**

Example

A $\bowtie_\theta$ B

Theta join can use any conditions in the selection criteria.

For example:

A $\bowtie_{A.column\ 2 > B.column\ 2}$ (B)

A ⋈ A.column 2 > B.column 2 (B)	
column 1	**column 2**
1	2

EQUI join:

When a theta join uses only equivalence condition, it becomes a equi join.

For example:

A $\bowtie_{A.column\ 2 = B.column\ 2}$ (B)

A ⋈ A.column 2 = B.column 2 (B)	
column 1	**column 2**
1	1

EQUI join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

C	
Num	**Square**

2	4
3	9

D	
Num	**Cube**
2	8
3	18

C ⋈ D

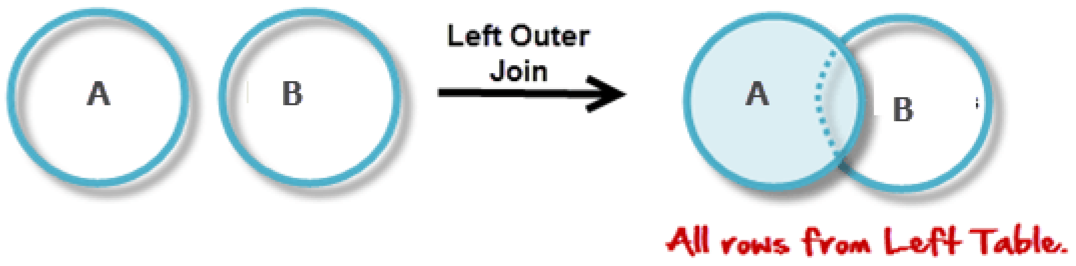
C ⋈ D		
Num	**Square**	**Cube**
2	4	4
3	9	9

OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join(A ⟕ B)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



Consider the following 2 Tables

A	
Num	**Square**
2	4
3	9
4	16

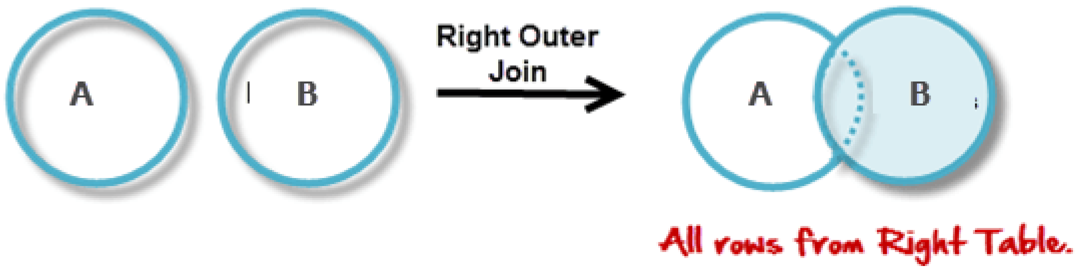
B	
Num	**Cube**
2	8
3	18
5	75

A ⟕ B

A ⋈ B		
Num	**Square**	**Cube**
2	4	4
3	9	9
4	16	-

Right Outer Join: (A ⟖ B)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.



A ⟖ B

A ⋈ B		
Num	**Cube**	**Square**
2	8	4
3	18	9
5	75	-

Full Outer Join: (A ⟗ B)

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

A ⟗ B

A ⋈ B		
Num	**Cube**	**Square**
2	4	8
3	9	18
4	16	-
5	-	75