R23

II B. Tech II Semester Regular Examinations, May/June - 2025

DATA BASE MANAGEMENT SYSTEMS

(Common to CSE, CSE (AIML, CS, DS), AI, IT)

Time: 3 hours MODEL QUESTION PAPER Max. Marks: 70

Note: 1. Question Paper consists of two parts (**Part-A** and **Part-B)**
2. Answer **ALL** the question in **Part-A**
3. Answer any **FIVE** Questions from **Part-B**

PART – A

1. A) List any two differences between a database and a file system. (10 X 2 = 20 M)
B) Mention two types of attributes in ER diagrams.
C) Define integrity constraints with examples.
D) What is the purpose of the ALTER TABLE command?
E) Write a basic SQL query using the SELECT and WHERE clauses
F) List two numeric functions available in SQL.
G) Define the Normalization?
H) List three advantages of normal forms.
I) Write the need of serializability in transaction management.
J) Differentiate primary index from secondary index.

PART - B

2. A) Explain the architecture of DBMS with a neat sketch. [10M]

Or

B) Compare and contrast various Data Models. [5M]
C) Explain the Generalization and Specialization with suitable ER Modeling. [5M]

3. A) Consider the Bank Management System. [10M]]

account (account number, branch_name, balance)
branch (branch name, branch_city, assets)
customer (customer name customer_street, customer_city)
loan (loan_number, branch_name, amount)
depositor ((customer_name, account_number)
borrower (customer_name, loan_number)
Answer the following queries using SQL Statements
1. List all branch names and their assests
2. List all accounts of Brooklyn branch
3. List all loans with amount > 1000.
4. List all accounts of Perryridge branch with balance < 1000.
5. List Numbers of accounts with balances between 700 and 900

OR

B) Explain the relational algebra opertions with suitable examples. [10M]

4.A) Explain the different Date,Numeric and String function with an example. [10M]

OR

B) Define the Join in SQL? And explain the types of joins with syntax and example. [5M]
C)Define the View in SQL? And explain the types of joins with syntax and example. [5M]

5.A) Define the Normalization and Explain the different normal forms? [10M]

OR

B) Elaborate the importance of computing closure of functional dependencies.

Explain the procedure with an example. [10M]

6.A) Define the transaction? Explain the properties of transactions. [5M]
B) Discuss about view serializability. [5M]

OR

C)Define the B+ Tree, discuss characteristics and operations supported by B+ Tree with suitable example. [10M]

Answers

1. A) List any two differences between a database and a file system.

Ans)Data Management and Querying:

- **File System**: Stores data in files and folders; retrieval requires manual parsing and handling by custom programs.
- **Database**: Manages data using structured query languages (like SQL), supporting efficient searching, filtering, and relationships between data.

Data Integrity and Security:

- **File System**: Limited mechanisms for ensuring data integrity and access control.
- **Database**: Enforces constraints (like primary keys, foreign keys) and supports fine-grained access control and transaction management to ensure data consistency and security.

B) Mention two types of attributes in ER diagrams.

Ans) There are different types of attributes as discussed below-

- Simple Attribute
- Composite Attribute
- Single-Valued Attribute
- Multi-Valued Attribute
- Derived Attribute
- Complex Attribute
- Stored Attribute
- Key Attribute
- Null Attribute

C) Define integrity constraints with examples.

Ans) The following constraints are commonly used in SQL:

1. NOT NULL - Ensures that a column cannot have a NULL value
2. UNIQUE - Ensures that all values in a column are different
3. PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
4. FOREIGN KEY - Prevents actions that would destroy links between tables
5. CHECK - Ensures that the values in a column satisfies a specific condition
6. DEFAULT - Sets a default value for a column if no value is specified

D)) What is the purpose of the ALTER TABLE command?

Ans) The **ALTER TABLE** command in SQL is used to **modify the structure of an existing table** in a database.

1.Adding a new column

```
SQL> Alter table table_name add (new_column datatype);
```

2.Modifying an existing column

SQL> Alter table table_name modify (old_column datatype(size));

3. Deleting a column

SQL> Alter table table_name drop column column_name;

E) Write a basic SQL query using the SELECT and WHERE clauses.

Ans) The **SELECT** statement in SQL is used to **retrieve data** from one or more tables in a database

Ans) SQL>select column_list from table_name where(condition);

F) List two numeric functions available in SQL.

Ans) 1)ABS(number) :- Returns the **absolute value** of a number.

SQL>SELECT ABS(-11) FROM dual;

2) CEIL(number) or CEILING(number) – Rounds a number up to the nearest integer.

SQL>SELECT ceil(5.67),floor(5.67) FROM dual;

3)POWER(x, y) – Returns x raised to the power of y.

SQL>SELECT power(3,2) FROM dual;

G) Define the Normalization?

Ans) Normalization, in the context of databases, is the process of organizing data to reduce redundancy and improve data integrity by breaking down large tables into smaller, more manageable tables and establishing relationships between them. This process helps minimize update, insertion, and deletion anomalies, making the database more flexible and consistent.

H) List out the advantages of normal forms.

Ans) **1. Reduced Data Redundancy:**

- Normalization minimizes the storage of duplicate data by storing each piece of information only once.
- This reduces storage space and improves efficiency.
- Updates only need to be made in one place, reducing the risk of inconsistencies.

2. Improved Data Integrity:

- Normalization enforces rules and constraints, ensuring data accuracy and consistency.
- It eliminates anomalies like update, insertion, and deletion anomalies, which can lead to data inconsistencies.
- Data is organized in a structured manner, making it easier to manage and maintain.

3. Enhanced Database Performance:

- Normalization optimizes database storage and reduces the need for full-table scans.
- This leads to faster query execution and improved overall database performance.
- It also simplifies database design and makes it easier to maintain and update.

4. **Easier Maintenance and Management:**

- A well-normalized database is easier to understand and maintain.
- Updates, insertions, and deletions are simplified, reducing the risk of errors.
- Data is more organized and consistent, making it easier to troubleshoot problems.

5. Improved Security:

- Normalization can help enforce access control and security measures.
- By limiting access to specific tables, it can enhance data security.

6. Enhanced Scalability and Flexibility:

- A normalized database can be more easily scaled to accommodate growing data volumes.
- It is also more flexible and can be adapted to changing business needs.

I) Write the need of serializability in transaction management.

Ans) Serializability is essential in transaction management to ensure the **correctness and consistency of the database** when multiple transactions are executed concurrently. It guarantees that the outcome of executing multiple transactions concurrently is the same as if the transactions were executed one after another in some serial order. Here's why it is needed:

1. **Maintains Database Consistency**: Ensures that concurrent transactions do not violate the consistency constraints of the database.
2. **Avoids Anomalies**: Prevents concurrency-related issues such as lost updates, temporary inconsistency, and uncommitted data being read (dirty reads).
3. **Preserves Isolation**: Serializability upholds the isolation property of ACID (Atomicity, Consistency, Isolation, Durability), ensuring each transaction appears isolated from others.
4. **Predictable Results**: Enables predictable and reproducible outcomes, critical for applications where data integrity is crucial.

J) Differentiate primary index from secondary index.

Ans)

Feature	**Primary Index**	**Secondary Index**
Definition	An index based on the **primary key** of a table.	An index based on **non-primary key** (non-unique) fields.
Uniqueness	Always **unique** (no duplicate values allowed).	May contain **duplicate values**.
Ordering	Data records are **physically ordered** based on it.	Does **not affect** the physical order of records.
Number per Table	Only **one** primary index per table.	**Multiple** secondary indexes can exist.
Access Efficiency	Typically **faster** for primary key lookups.	May be **slower**, especially with duplicate entries.
Implementation	Often implemented as a **sparse index**.	Typically implemented as a **dense index**.
Use Case	Used when searching using **primary key values**.	Used for queries on **non-key attributes**.

2.A) Explain the architecture of DBMS with a neat sketch.

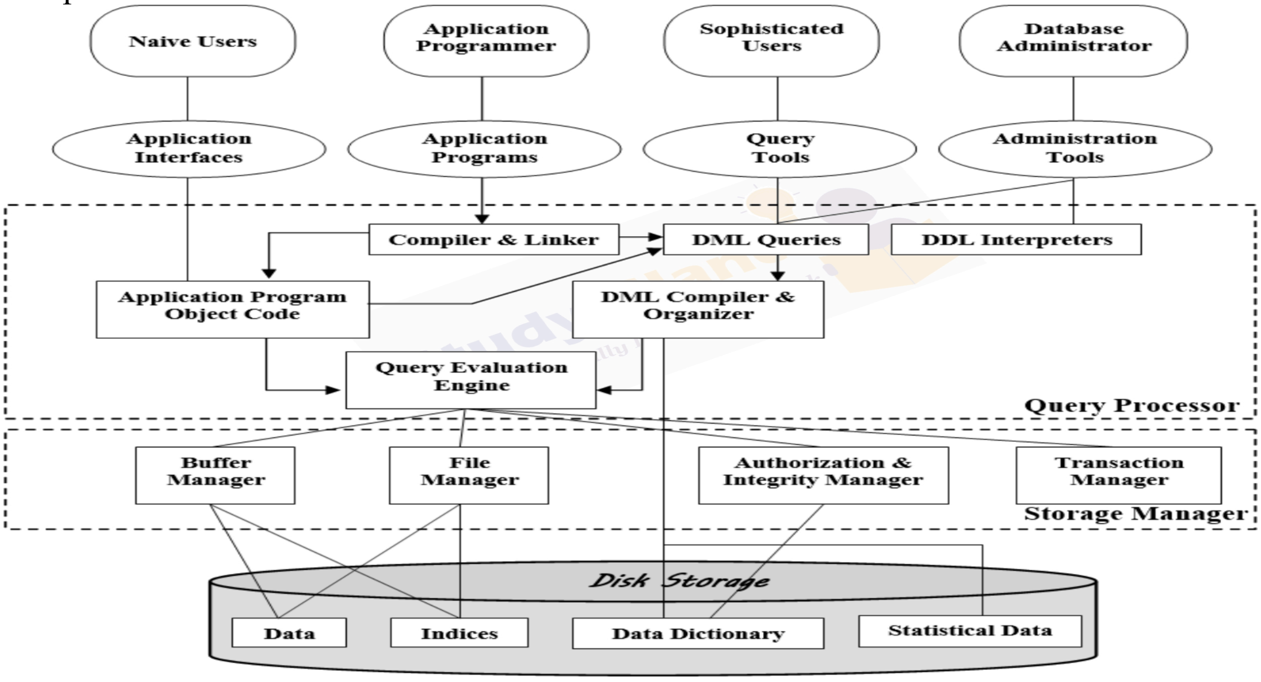
Ans) Database System Architecture

The typical structure of DBMS is based on Relational data model.

The top part of the architecture shows application interfaces used by naive users, application programs created by application programmers, query tools used by sophisticated users and administration tools used by database administrator.

The lowest part of the architecture is for disk storage.

The Middle two parts(**Query processor and storage manager**) are important components of database architecture.



The interactive query processor helps the database system to simplify and facilitate access to data. It consists of DDL(**Data Definition Language**) interpreter, DML(**Data Manipulation Language**) compiler and query evaluation engine.

The following are various functionalities and components of query processor

- **DDL interpreter**: This is basically a translator which interprets the DDL statements in data dictionaries.
- **DML compiler**: It translates DML statements query language into an evaluation plan. This plan consists of the instructions which query evaluation engine understands.
- **Query evaluation engine**: It executes the low-level instructions generated by the DML compiler.

When a user issues a query, the parsed query is presented to a query optimizer, which uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan is a blueprint for evaluating a query. It is evaluated by query evaluation engine.

In database architecture, the **Query Processor** is responsible for interpreting and executing database queries. It acts as a bridge between the user's query and the database engine, ensuring that the query is understood, optimized, and executed efficiently. The main components of a Query Processor are:

1. DML Compiler (Data Manipulation Language Compiler)

- **Function:** Translates high-level queries (like SQL) into a lower-level language that the database engine can understand.
- **Process:** Checks for syntactic and semantic correctness before translating the query into an internal form, such as a query tree or query graph.

2. Query Parser

- **Function:** Parses the query to check for syntax errors and to verify the query structure.
- **Process:**
    - **Syntax Analysis:** Ensures the query follows the grammatical rules of the query language.
    - **Semantic Analysis:** Checks for logical errors, such as verifying if the specified tables and columns exist.

3. Query Optimizer

- **Function:** Optimizes the query by finding the most efficient execution plan.
- **Process:**
    - Evaluates different query execution plans.
    - Considers factors like indexing, join methods, and data distribution.
    - Chooses the optimal plan based on cost estimation (e.g., CPU usage, memory, and I/O operations).

4. Query Plan Generator

- **Function:** Generates a sequence of operations to execute the query.
- **Process:** Converts the optimized query into a sequence of low-level instructions that the query executor can understand.

5. Query Executor : Executes the query plan and retrieves the required data.

- **Process:** Performs operations like scanning, joining, sorting, and filtering.
- Communicates with the storage engine to access or update the data.

6. Runtime Database Manager : Coordinates between the query executor and the storage manager.

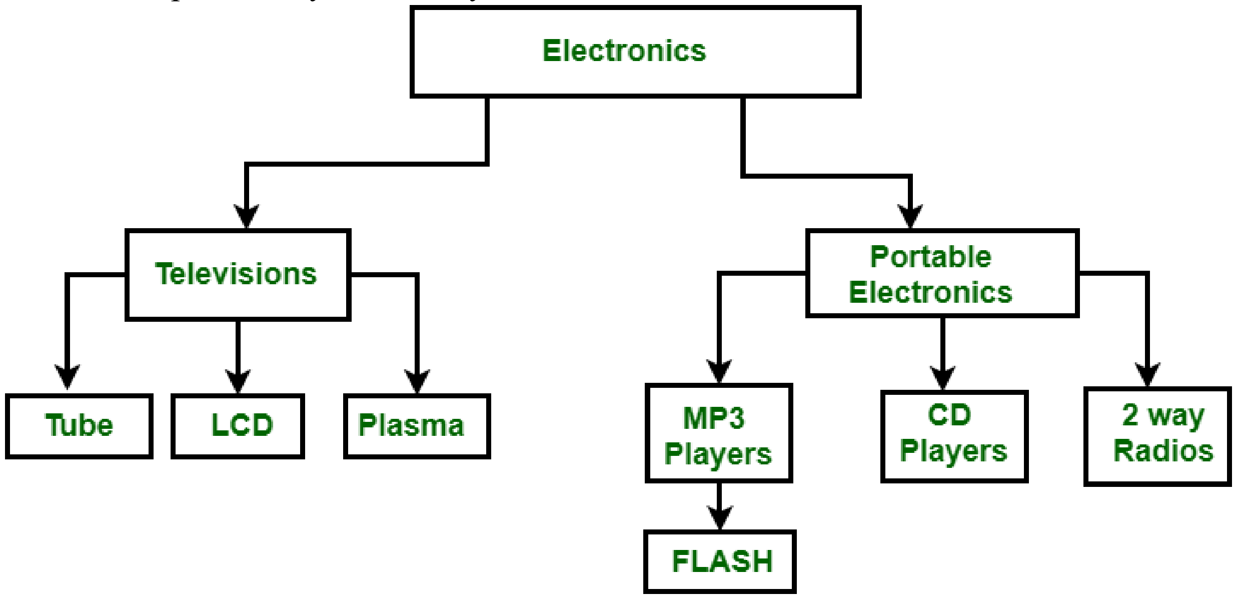
- **Process:** Ensures data integrity and consistency during the query execution process.

In database architecture, the **Storage Manager** is responsible for handling the storage, retrieval, and update of data in the database. It acts as an interface between the low-level data stored on physical storage devices and the higher-level query processor components. Its primary role is to efficiently manage data storage, access, and integrity.

2.B) Compare and contrast various Data Models.
In order to define the connections, organization, and structure of data in a database management system (DBMS), data models are essential. The effectiveness of a database's ability to store, retrieve, and alter data is greatly influenced by the choice of data model. The Hierarchical, Network, and Relational models are some of the oldest types of data models; each has special traits and applications

What is a Hierarchical Data Model?
The hierarchical data model is the oldest type of the data model. It was developed by IBM in 1968. It organizes data in a tree-like structure. Hierarchical model consists of the following :

- It contains nodes which are connected by branches.
- The topmost node is called the root node.
- If there are multiple nodes appear at the top level, then these can be called root segments.
- Each node has exactly one parent.
- One parent may have many children.



In the above figure, Electronics is the root node which has two children i.e. Televisions and Portable Electronics. These two has further children for which they act as parent. For example: Television has children as Tube, LCD and Plasma, for these three Television act as parent. It follows one to many relationship.

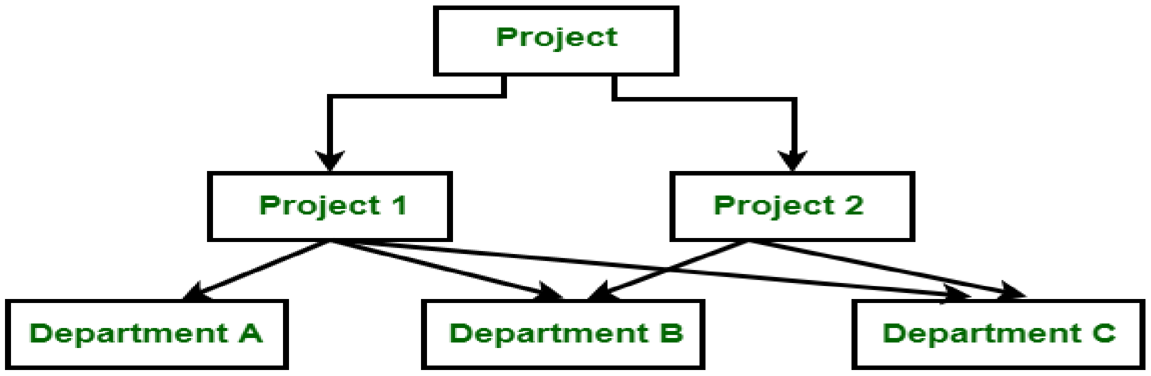
Advantages of the Hierarchical Data Model

- Because of its tree form, it is easy to grasp.
- Retrieving data in a one-to-many connection is efficient.

Disadvantages of the Hierarchical Data Model

- Inflexibility in reorganizing data.
- accessing complicated data structures may be challenging.
- redundant data storage, which might cause anomalies and inconsistencies.

What is a Network Data Model?
It is the advance version of the hierarchical data model. To organize data it uses directed graphs instead of the tree-structure. In this child can have more than one parent. It uses the concept of the two data structures i.e. Records and Sets.



In the above figure, Project is the root node which has two children i.e. Project 1 and Project 2. Project 1 has 3 children and Project 2 has 2 children. Total there are 5 children i.e Department A, Department B and Department C, they are network related children as we said that this model can have more than one parent. So, for the Department B and Department C have two parents i.e. Project 1 and Project 2.

Advantages of the Network Data Model

- Because of its numerous parent ties, it is more adaptable than the hierarchical approach.
- Ideal for managing intricate, many-to-many connections.

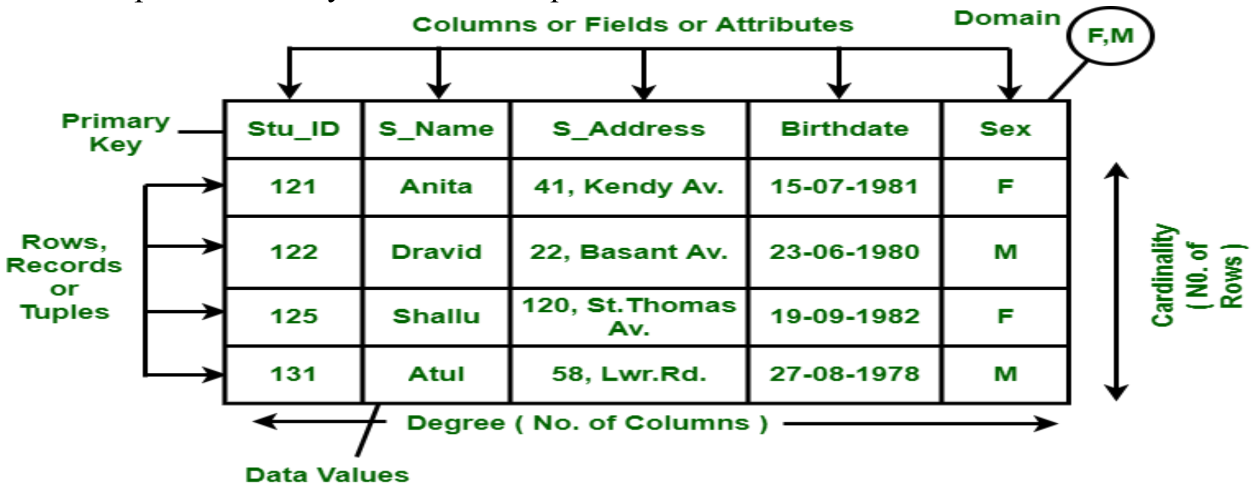
Disadvantages of the Network Data Model

- Increased complexity in database design and management.
- requires complex programming in order to manage and work with data.

What is a Relational Data Model?

The relational data model was developed by E.F. Codd in 1970. There are no physical links as they are in the hierarchical data model. Following are the properties of the relational data model :

- Data is represented in the form of table only.
- It deals only with the data not with the physical structure.
- It provides information regarding metadata.
- At the intersection of row and column there will be only one value for the tuple.
- It provides a way to handle the queries with ease.


Columns or Fields or Attributes
Domain
F,M
Primary Key
Rows, Records or Tuples
Cardinality (NO. of Rows)
Degree (No. of Columns)
Data Values

Stu_ID	S_Name	S_Address	Birthdate	Sex
121	Anita	41, Kendy Av.	15-07-1981	F
122	Dravid	22, Basant Av.	23-06-1980	M
125	Shallu	120, St.Thomas Av.	19-09-1982	F
131	Atul	58, Lwr.Rd.	27-08-1978	M

Advantages of the Relational Data Model

- High data independence and flexibility.
- offers robust and user-friendly querying features.
- removes duplication by use of normalization.

Disadvantages of the Relational Data Model

- For certain kinds of straightforward data retrieval tasks, they may not perform

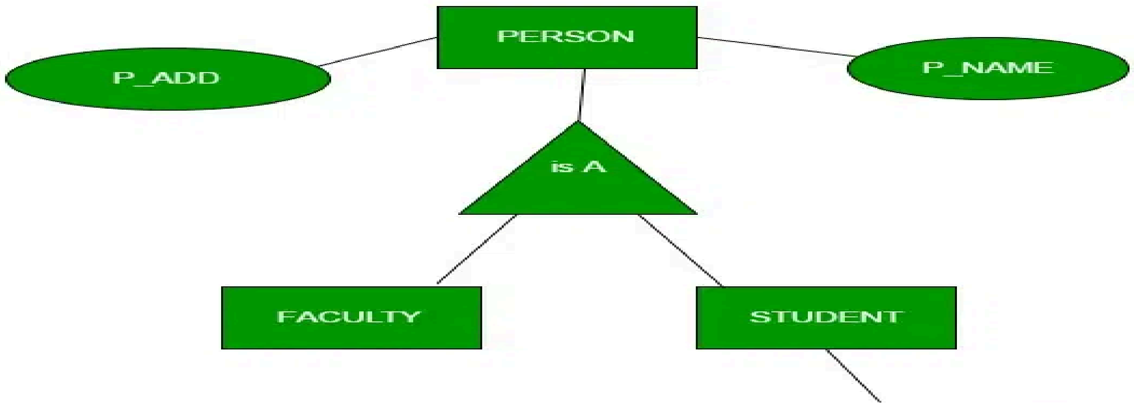
as well as hierarchical models.
- demands a deeper comprehension of SQL and normalization principles.

2.C) Explain the Generalization and Specialization with suitable ER Modeling.

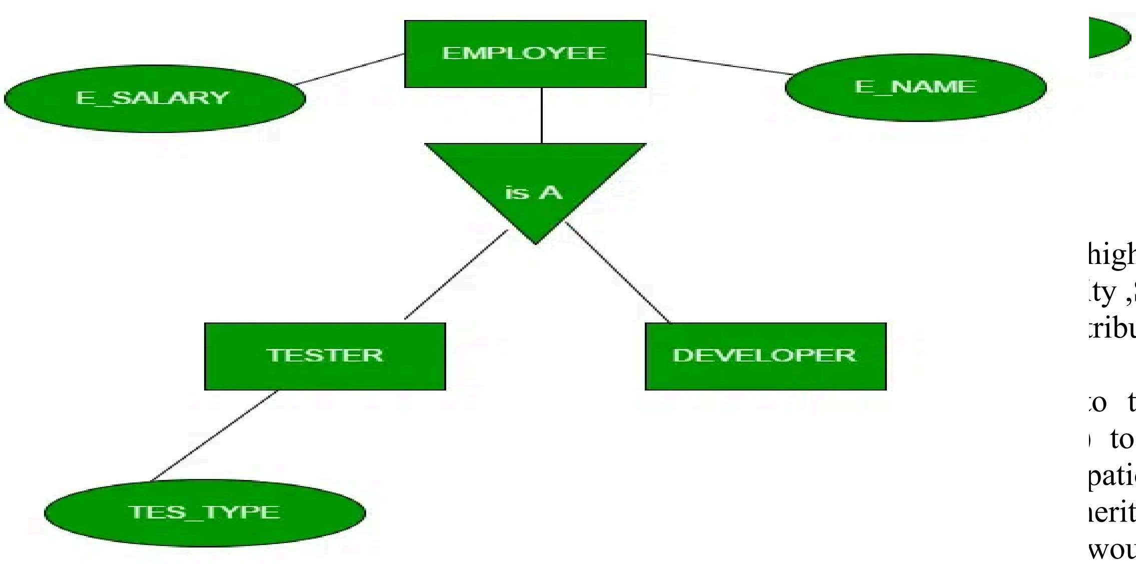
Ans) **Generalization**

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher-level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher-level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, and P_ADD become part of a higher entity (PERSON), and specialized attributes like S_FEE become part of a specialized entity (STUDENT).

Generalization is also called as ‘ Bottom-up approach”.



S
I
t
l
c
c
a
(



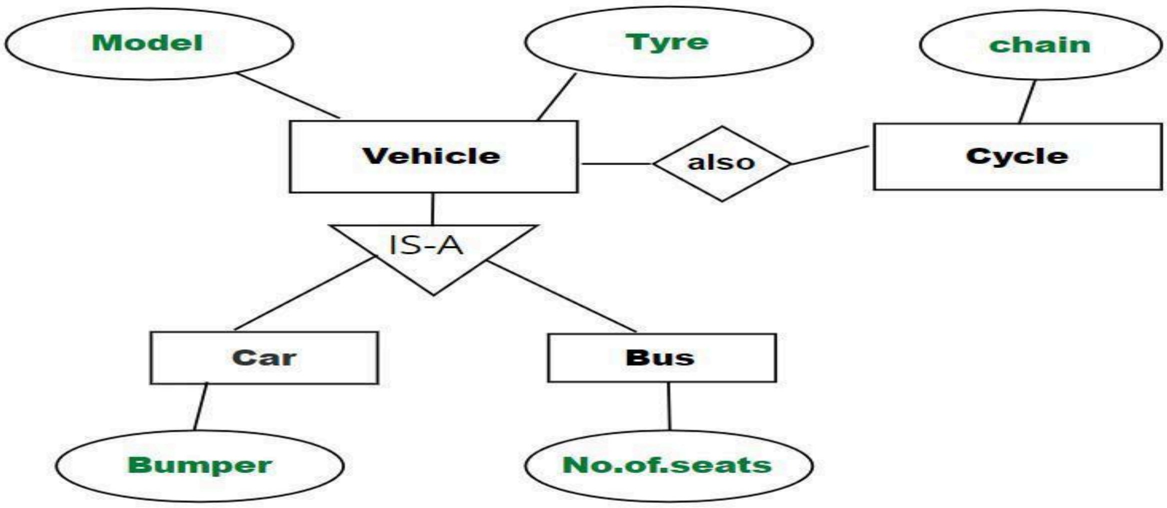
Specialization

I

higher
ty ,So
ribute

o the
to a
pation
herited
would
pation

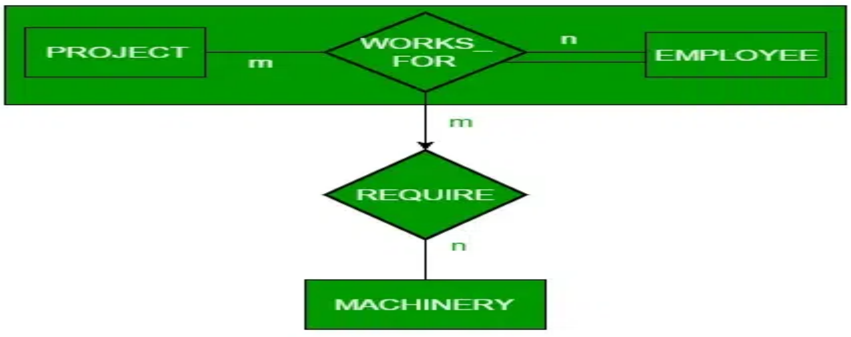
inheritance only refers to the inheritance of participation constraints, not the actual relationships between entities.



Example of Relation

Aggregation

An ER diagram is not capable of representing the relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher-level entity. Aggregation is an abstraction through which we can represent relationships as higher-level entity sets. For Example, an Employee working on a project may require some machinery. So, REQUIRE relationship is needed between the relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into a single entity and relationship REQUIRE is created between the aggregated entity and MACHINERY.



Aggregation

3.A)

1.List all branch names and their assests2.
2.List all accounts of Brooklyn branch
3.List all loans with amount > 1000.
4.List all accounts of Perryridge branch with balance < 1000.
5.List Numbers of accounts with balances between 700 and 900

Ans) 1. List all branch names and their assets:
```
SQL> SELECT branch_name, assets FROM branch;
```
2. List all accounts of Brooklyn branch:
```
SQL>SELECT account_number, branch_name, balance
              FROM account
              WHERE branch_name= 'Brooklyn';
```
3. List all loans with amount > 1000:
```
SQL>SELECT loan_number, branch_name, amount  FROM loan
              WHERE amount > 1000;
```
4. List all accounts of Perryridge branch with balance < 1000:
```
SQL>SELECT account_number, branch_name, balance FROM account
```

```
              WHERE branch_name = 'Perryridge' AND balance < 1000;
    5. List numbers of accounts with balances between 700 and 900:
           SQL>SELECT account_number FROM account
            WHERE balance BETWEEN 700 AND 900;
```

3.B) Explain the relational algebra opertions with suitable examples.

Ans) Relational Algebra is a procedural query language used in relational databases. It provides a set of operations to manipulate and retrieve data from relational tables.

Types of Relational Algebra Operators

Relational Algebra has two main types of operators:

1. **Set Operators** – Perform operations on sets of tuples (rows).
2. **Relational Operators** – Operate on relations (tables) to filter or combine data.

a) Selection (σ) – Filters Rows

- Selects rows (tuples) that satisfy a given condition.
- Equivalent to WHERE in SQL.

Notation: σcondition(Relation)σ_{condition} (Relation)σcondition(Relation)
Example: Find employees with a salary greater than 50000.
Ex : σsalary>50000(Employees)

b) Projection (π) – Selects Columns

- Selects specific columns (attributes) from a table.
- Equivalent to SELECT column_names in SQL.

Notation: πcolumn1,column2,...(Relation)
Example: Get only employee names and salaries.
Ex : πempname,salary(Employees)

c) Cartesian Product (×) – Combines Relations

- Combines each row of one relation with every row of another.
- Forms the basis for **JOIN** operations in SQL.

Notation: Relation1×Relation2
Example: Get all possible pairs of employees and departments.
Ex: Employees×Departments

d) Union (∪) – Combines Tuples from Two Relations

- Combines two tables with **same attributes** and removes duplicates.
- Equivalent to UNION in SQL.

Notation: Relation1∪Relation2
Example: Get a list of all employees and managers.
Ex : Employees∪Managers

e) Set Difference (−) – Finds Tuples in One Relation but Not in Another

- Retrieves rows in one table that do not exist in another.
- Equivalent to MINUS in Oracle SQL.

Notation: Relation1−Relation2
Example: Find employees who are **not** managers.
Ex : Employees−Managers

f) Intersection (∩) – Finds Common Tuples Between Relations

- Retrieves only the common rows between two tables.

- Equivalent to INTERSECT in SQL.

Notation: Relation1∩Relation2
Example: Find employees who are also managers.
Employees∩Managers
2. Joins in Relational Algebra
Joins are special types of Cartesian Product used to combine related tables.
a) Natural Join (⋈) – Combines Related Tables Automatically

- Joins tables based on **common attributes (column names and values)**.
- Equivalent to NATURAL JOIN in SQL.

Notation: Relation1⋈Relation2
Example: Combine employees with their department information.
Ex : Employees⋈Departments

b) Theta Join (⋈θ) – Joins Tables with a Condition

- Joins tables based on a specified condition (θ).
- Equivalent to INNER JOIN in SQL.

Notation: Relation1 $⋈_{condition}$Relation2
Example: Join employees and departments where dept_id matches.
Employees$⋈_{Employees.deptid=Departments.deptidDepartments}$Employees
c) Left Outer Join (⟕) – Keeps All Left Table Rows

- Returns all rows from the **left table** and matching rows from the **right table**.

Notation: Relation1⟕Relation2
d) Right Outer Join (⟖) – Keeps All Right Table Rows

- Returns all rows from the **right table** and matching rows from the **left table**.

Notation: Relation1⟖Relation2
e) Full Outer Join (⟗) – Keeps All Rows from Both Tables

- Returns all rows when there is a match, and NULLs where there is no match.

Notation: Relation1⟗Relation2

4.A) Explain the different Date,Numeric and String function with an example.
Dual Table :- Dual table is a dummy table, It having only one row and one column, Value of the dual table is ‘x’ . By using dual table we can perform arithmetic Operations and can use date retrieval functions.
A) Date Functions:- SQL supports the following date functions.
1) sysdate :- sysdate returns the today’s sys date value.
Syntax : SQL> select sysdate from dual;

```
SYSDATE
---------
28-FEB-25
```

2)sysdate + Number :- it returns the date value after adding some number of days.
SQL> select sysdate,sysdate+90 from dual;

```
SYSDATE   SYSDATE+9
--------- ---------
28-FEB-25 29-MAY-25
```

3)last_day(date) :- It returns the last day of the month.

```
        SQL> select sysdate,last_day(sysdate) from dual;
                SYSDATE   LAST_DAY
                ---------       ---------
                28-FEB-25   28-FEB-25
```

4)next_day(date):- It returns next date value by passing next day value.

```
SQL> select sysdate,next_day(sysdate,'Friday') from dual;

                SYSDATE   NEXT_DAY
                ---------       ---------
                28-FEB-25    07-MAR-25
```

5)months_between(Date1,Date2):- It returns number of months between the two dates.

```
SQL> select months_between(sysdate,'28-Feb-24') from dual;

MONTHS_BETWEEN(SYSDATE,'28-FEB-24')
-----------------------------------------------------------
                          12
```

B)Numeric Functions :-

1)abs(value):- It returns the absolute value.

```
SQL> SQL> select abs(-11), abs(12) from dual;
                 ABS(-11)    ABS(12)
                ----------       ----------
                     11         12
```

2)mod(m,n):- It calculate mod value that is remainder value passed by the function.

```
        SQL> select mod(20,3) from dual;
         MOD(20,3)
        ----------
                2
```

3)power(m,n):-It calculates power of a number.

```
SQL> select power(3,2) from dual;
        POWER(3,2)
        ----------
                9
```

4)ceil(value):- It returns the ceil value passed by the function.

5)floor(value):- It returns the floor value passed by the function.

```
SQL> select ceil(5.67), floor(5.67) from dual;
                CEIL(5.67)        FLOOR(5.67)
                ----------             -----------
                      6              5
```

6)tan(value):- It calculate trigonometric value pass by the function. It calculate radians value.

```
SQL> select tan(45) from dual;
          TAN(45)
```

```
        ----------
        1.61977519
```
7)sqrt(value):- It calculate the square root of a number.

```
SQL> select sqrt(25) from dual;
          SQRT(25)
        ----------
                5
```
8) SQL> select least(11,22),greatest(11,22) from dual;

```
        LEAST(11,22)        GREATEST(11,22)
        ------------                ---------------
              11                    22
```

C)String Functions :-

1)InitCap('String'):- The InitCap() function in SQL is used to capitalize the first letter of each word in a string while converting the rest of the letters to lowercase.

```
SQL> select InitCap('my name is naresh') from dual;
                INITCAP('MYNAMEIS
                -----------------
                My Name Is Naresh
```
2)Length('String'):- The LENGTH() function in SQL is used to return the number of characters in a string.

```
SQL> select length('Srinivas') from dual;
        LENGTH('SRINIVAS')
        ------------------
                  8
```

3)upper('String'):- The UPPER() function in SQL converts all characters in a given string to uppercase. It is useful for case-insensitive comparisons and formatting text data.

```
SQL> select upper('srinivasarao') from dual;
              UPPER
        -----------------------
        SRINIVASARAO
```
4)lower('String'):- The LOWER() function in SQL converts all characters in a given string to lowercase. It is useful for case-insensitive comparisons and standardizing text data.

```
SQL> select lower('NARESH') from dual;
        LOWER
        ----------
        naresh
```

5)substr('String',starting character number,number of characters):- The SUBSTR() function in SQL extracts a portion of a string based on a specified starting position and length. It is commonly used for string manipulation.

```
        SQL> select substr('SRINIVAS',4,5) from dual;
        SUBST
        -----
        NIVAS
```
6)ltrim('String',Number of characters in a string','special character'):- The LTRIM() function in SQL removes leading spaces (spaces from the left) from a

string. It is useful for cleaning up text data before processing or comparison.

```
SQL> select LTRIM('  S RINIVAS') from dual;
        LTRIM
        ---------
        S RINIVAS
```

7)rtrim('String',Number of characters in a string','special character'):- The RTRIM() function in SQL removes trailing spaces (spaces from the right) from a string. It is useful for cleaning up text data before processing or comparison.

```
SQL> select RTRIM('SRINIVAS   ') from dual;
        RTRIM
        --------
        SRINIVAS
```

8)concat('String1','String2'):- The CONCAT() function in SQL is used to combine two or more strings into one. It is useful for merging text values in queries.

```
SQL> select concat('Soft','Ware') from dual;
        CONCAT
        --------
        SoftWare
```

9)trim('String'):- The TRIM() function in SQL removes leading, trailing, or both leading and trailing spaces (or other specified characters) from a string. It is useful for cleaning up text data.

```
SQL> select trim('  srinivasa rao   ') from dual;
                TRIM
                -------------
                srinivasa rao
```

10)ASCII('character'):- The ASCII() function in SQL returns the **ASCII (American Standard Code for Information Interchange) value** of the first character in a given string. It is useful for checking character codes, sorting, and data validation.

```
SQL> select ASCII('A')  , ASCII('a')  from dual;
        ASCII('A') ASCII('A')
        ---------- ----------
              65        97
```

11)Replace('String',Replaced character',Replacing Characters'):- The REPLACE() function in SQL is used to replace all occurrences of a specified substring within a string with another substring. It is useful for modifying text data, cleaning up strings, or standardizing values.

```
SQL> select Replace('Back','B','Bl')  from dual;
        REPLA
        -----
        Black
```

12)Lpad():-The LPAD() function in SQL is used to **left-pad** a string with a specified character until it reaches a certain length. It is useful for formatting

text, aligning values, and ensuring fixed-length outputs.
13)Rpad():-The RPAD() function in SQL is used to **right-pad** a string with a specified character until it reaches a certain length. It is useful for formatting text, ensuring fixed-length values, and aligning data.

```
SQL> select LPAD('SRINIVAS',12,'*'),RPAD('SRINIVAS',12,'*')  from dual;

        LPAD                          RPAD
        -------------------                   ------------
        ****SRINIVAS                  SRINIVAS****
```

4.B) Define the Join in SQL? And explain the types of joins with syntax and example.

Ans) Ans) A JOIN clause is used to combine rows from two or more tables, based on a related column between them. SQL Supports the following jojns.
1)**CROSS JOIN** :- A cross join is a type of join that returns the Cartesian product of rows from the tables in the join. In other words, it combines each row from the first table with each row from the second table.
Syntax :

```
SQL> Select table1.column1,table1.column2, table2.column1, table2.column2
    from table1,table2;
```

2)EQUI JOIN:- Returns records that have matching values in both tables
Syntax :

```
SQL> Select table1.column1,table1.column2, table2.column1,
        table2.column2 from table1,table2
        where table1.column= table1.column;
```

3.A)LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
Syntax :

```
SQL> Select table1.column1,table1.column2, table2.column1,
        table2.column2 from table1,table2
        where table1.column= table1.column(+);
```

3.B)RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table.
Syntax :

```
SQL> Select table1.column1,table1.column2, table2.column1,
        table2.column2 from table1,table2
        where table1.column(+)= table1.column;
```

3.C)FULL (OUTER) JOIN:- Returns all records when there is a match in either left or right table
Syntax :

```
SQL> Select table1.column1,table1.column2, table2.column1,
        table2.column2 from table1,table2
        where table1.column= table1.column(+)
        Union
        Select table1.column1,table1.column2, table2.column1,
        table2.column2 from table1,table2
```

where table1.column(+)= table1.column;

4)Self Join :-A self join is a type of join operation in a relational database where a table is joined with itself. It allows you to combine rows from the same table based on a related condition.

Syntax :

SQL>SELECT *column_name(s)* FROM *table1 T1, table1 T2*
WHERE *condition*;

5.A) Define the Normalization and Explain the different normal forms?

Ans) **Types of Normal Forms**

First Normal Form (1NF): This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name.

1)Data Must be in a Tabular form

2) Intersection of row and column should have atomic value.

Student Number	SName	Address	Courses
1	Naresh	NRT	C, C++, JAVA
2	Suresh	Guntur	Oracle, MySQL

Fig: Unnormalized Relation

Student Number	SName	Address	Courses
1	Naresh	NRT	C
1	Naresh	NRT	C++
1	Naresh	NRT	JAVA
2	Suresh	Guntur	Oracle
2	Suresh	Guntur	MySQL

Fig : First Normal Form

Second Normal Form (2NF): 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.

1)Data Must be in a Tabular form

2)It Should be in **First Normal Form**

3)It Removes Partial Function Dependencies

EmpNo	EName	DOJ	Deptno	DName	Location
1	Naresh	10-Jan-25	10	CSE	Bangalore
2	Suresh	20-Jan-25	10	CSE	Bangalore
3	Ramesh	25-Jan-25	20	ECE	HYD
4	Mahesh	30-Jan-25	20	ECE	HYD

Fig : First Normal Form

EmpNo	EName	DOJ	Deptno
1	Naresh	10-Jan-25	10
2	Suresh	20-Jan-25	10
3	Ramesh	25-Jan-25	20
4	Mahesh	30-Jan-25	20

Deptno	DName	Location
10	CSE	Bangalore

20	ECE	HYD

Fig : Second Normal Form

Third Normal Form (3NF): 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.
1)Data Must be in a Tabular form
2)It Should be in **Second Normal Form**
3)It Removes Transitive Function Dependencies

StudentID	SName	Course	PINCode	Name of the Town
1	Naresh	Java	522 601	Narasaraopet
2	Suresh	Oracle	500 001	Hyderabad

Fig : Second Normal Form

StudentID	SName	Course	PINCode
1	Naresh	Java	522 601
2	Suresh	Oracle	500 001

PINCode	Name of the Town
522 601	Narasaraopet
500 001	Hyderabad

Fig : Third Normal Form

Boyce-Codd Normal Form (BCNF): BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.
For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:

1. It should be in the **Third Normal Form**.
2. for any dependency A → B, A should be a **super key**.

The second point sounds a bit tricky, right? In simple words, it means, that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**.
Below we have a college enrolment table with columns student_id,subject and Professor.

student_id	**subject**	**professor**
101	Java	Naresh
101	C++	Suresh
102	Java	Mahesh
103	C#	Rajesh
104	Java	Ramesh

In the table above:

- One student can enrol for multiple subjects. For example, student with **student_id** 101, has opted for subjects - Java & C++
- Each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like we have for Java.

What do you think should be the Primary Key?

Well, in the table above student_id, subject together form the primary key, because using student_id and subject, we can find all the columns of the table.
One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.
Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

Fourth Normal Form (4NF): 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.

A database table is in Fourth Normal Form (4NF) if it's in Boyce-Codd Normal Form (BCNF) and has no multivalued dependencies (MVDs) that are not trivial or dependent on a candidate key.
Consider a table StudentCoursesHobbies with columns StudentID, Course, and Hobby.

StudentID	Course	Hobby
S1	Math	Painting
S1	Physics	Painting
S1	Math	Playing
S1	Physics	Playing
S2	Chemistry	Reading

In this table:

- **StudentID determines Course and Hobby:** A student can take multiple courses and have multiple hobbies.
- **There is a multivalued dependency:** For a single student (StudentID), there are multiple courses and hobbies.

- **This table is not in 4NF:** The multivalued dependencies (StudentID ->-> Course, StudentID ->-> Hobby) are not trivial and not dependent on a candidate key.

To achieve 4NF, decompose the table into two tables:

- **StudentCourses:** StudentID, Course
- **StudentHobbies:** StudentID, Hobby

What is Multi-valued Dependency?

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

Fifth Normal Form (5NF):

Fifth Normal Form (5NF), also known as Projection-Join Normal Form (PJ/NF), is the highest level of database normalization, addressing join dependencies to minimize redundancy and anomalies, especially when dealing with multi-valued facts and relationships.

5NF eliminate redundancy and ensure data integrity by breaking down tables into smaller, more manageable tables based on join dependencies.

Join Dependency: A table exhibits a join dependency if it can be reconstructed by joining two or more tables, each containing a subset of the original table's attributes.

Example:

Consider a scenario where agents represent companies, companies make products, and agents sell products. You might want to record which agent sells which product for which company. This can be modeled with a table that has attributes like AgentID, CompanyID, and ProductID. A 5NF design would ensure that this information is stored efficiently without redundancy.

AgentID	AgentName	CompanyID	CompanyName	Location	ProductID	ProductName	Price

Benefits:

- o Minimizes redundancy.
- o Enhances data integrity.
- o Simplifies data retrieval.

Drawbacks:

- Can lead to a larger number of tables and relationships, potentially making the database design more complex.
- May require more complex queries.

Q) Types of Functional Dependencies?

1. Partial Dependency: A partial dependency occurs when a non-key attribute (an attribute that's not part of any candidate key) depends on only a part of a composite primary key (a primary key made up of multiple attributes).

Example:

StudentID	CourseID	StudentName

Consider a table with a composite primary key (StudentID, CourseID). If StudentName depends only on StudentID, not on the combination of StudentID and CourseID, then StudentName has a partial dependency.

Impact: Partial dependencies lead to data redundancy and anomalies, which are problems that can occur when inserting, updating, or deleting data.

2. Full Dependency (Fully Functional Dependency): A full dependency exists when a non-key attribute depends on the entire primary key, not just a part of it.

StudentID	CourseID	StudentName	DOB	CONTACT

In the same (StudentID, CourseID) example, if Grade depends on both StudentID and CourseID, then Grade has a full dependency on the primary key.

Impact: Full dependencies are desirable for maintaining data integrity.

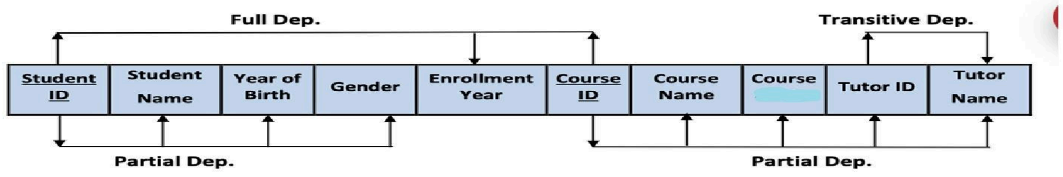
3. Transitive Dependency: A transitive dependency occurs when a non-key attribute depends on another non-key attribute, rather than directly on the primary key. If A → B and B → C, then A → C is a transitive dependency.

If StudentID → City and City → ZipCode, then StudentID → ZipCode is a transitive dependency.

StudentID	City	ZipCode

Impact: Transitive dependencies also lead to data redundancy and anomalies.

Identifying Full, partial and transitive dependencies:



Functional Dependency Diagram:

Full Dep:	(StudentID, CourseID) ➔ Enrollment_Year
Partial Dep:	StudentID ➔ Student_Name, Year_of_Birth, Gender
Partial Dep:	CourseID ➔ Course_Name, Course_Units, TutorID, Tutor_Name
Transitive Dep:	TutorID ➔ Tutor_Name

5.B) Elaborate the importance of computing closure of functional dependencies.
Explain the procedure with an example.

The **closure of functional dependencies (FDs)** is a fundamental concept in relational database design. It plays a crucial role in **normalization**, **candidate key identification**, and **checking for lossless decomposition**.

Why is Computing Closure Important?

1. **To Find Candidate Keys**
   By computing the closure of attribute sets, we can identify which sets of attributes uniquely determine all attributes of a relation.
2. **Normalization**
   Helps to detect **redundancies** and aids in decomposing relations into **normal forms** (like 2NF, 3NF, BCNF).
3. **Testing FD Implication**
   Closure helps determine whether a particular functional dependency **follows logically** from a given set of dependencies.
4. **Checking Equivalence of FD Sets**
   If two sets of FDs yield the same closure, they are considered equivalent.

Procedure to Compute Closure of an Attribute Set (X^+)

Let **X** be a set of attributes, and **F** be a set of functional dependencies. The **closure** of X with respect to F, denoted as **X^+**, is the set of all attributes that are functionally determined by X using F.

Steps:

1. **Start with X^+ = X.**
2. **Repeat** until no new attributes can be added:
   - For each FD A→BA \rightarrow BA→B in F:
     - If A⊆X+A \subseteq X^+A⊆X+, then add B to X^+.

Example : Given:
Relation R(A, B, C, D, E)
Set of FDs, F:

1. $A \rightarrow B$
2. $B \rightarrow C$
3. $A \rightarrow D$

4. D → E

Compute A^+ (Closure of A):

Step 1: Start with $A^+ = \{A\}$
Step 2:

- $A \rightarrow B \Rightarrow$ Add B $\Rightarrow A^+ = \{A, B\}$
- $B \rightarrow C$ (since $B \in A^+$) $\Rightarrow$ Add C $\Rightarrow A^+ = \{A, B, C\}$
- $A \rightarrow D \Rightarrow$ Add D $\Rightarrow A^+ = \{A, B, C, D\}$
- $D \rightarrow E$ ($D \in A^+$) $\Rightarrow$ Add E $\Rightarrow A^+ = \{A, B, C, D, E\}$

Result: $A^+ = \{A, B, C, D, E\}$
This means A is a **candidate key** for R, as it determines all attributes.

6.A) Define the transaction? Explain the properties of transactions.

Ans) a **transaction** is a sequence of one or more operations (such as read, write, update, or delete) performed as a single logical unit of work. A transaction ensures that a database remains in a consistent state even in cases of system failures or concurrent access by multiple users.

Transactions are essential in database management systems (DBMS) to maintain data integrity and consistency.

Properties of Transactions (ACID Properties):

Transactions are governed by four key properties known as **ACID**:

1. **Atomicity**:
   - Ensures that all operations within a transaction are completed successfully. If any part of the transaction fails, the entire transaction is rolled back.
   - Think of it as **"all or nothing"**.
2. **Consistency**:
   - Ensures that a transaction transforms the database from one **valid state** to another valid state.
   - Data integrity constraints must be maintained before and after the transaction.
3. **Isolation**:
   - Ensures that multiple transactions executing concurrently do not interfere with each other.
   - The intermediate state of a transaction should be **invisible** to other transactions.
   - Helps prevent problems like **dirty reads**, **non-repeatable reads**, and **phantom reads**.
4. **Durability**:

  - Ensures that once a transaction is **committed**, its changes are **permanent**, even in the event of a system crash.
  - Committed data is stored in **non-volatile memory**.

6.B) Discuss about view serializability in transaction management

Ans) View Serializability plays an essential role in **transaction management**, especially in **concurrent execution** of transactions in a **database system**. It ensures that interleaved transaction executions do not violate the correctness of the database.

Purpose in Transaction Management:

In transaction management, **schedules** define the order of operations from multiple transactions. Since executing transactions one-by-one (serially) reduces performance, **concurrent execution** is preferred for better system throughput.

However, concurrency can lead to **inconsistencies**. To avoid this, the DBMS uses **serializability criteria** to ensure that the **interleaved schedule behaves like a serial one** in terms of database correctness.

View Serializability:

1. **Schedule**: A sequence of read and write operations from one or more transactions.
2. **Serial Schedule**: Transactions are executed one after the other without overlapping.
3. **View Equivalent Schedule**: A non-serial schedule is said to be view equivalent to a serial schedule if:
   - Initial reads are the same.
   - Reads in both schedules read the same values.
   - Final writes are done by the same transactions.
4. **View Serializable Schedule**: A schedule that is view equivalent to a serial schedule.

Importance in Transaction Management:

- Offers a **more flexible** and **general form** of serializability.
- Helps in understanding which schedules are logically correct.
- Used in **theoretical analysis** of transaction systems and in **optimistic concurrency control** techniques.

6.C) Define the B+ Tree, discuss characteristics and operations supported by B+ Tree with suitable example.

Ans) A **B+ Tree** is an advanced type of **self-balancing tree data structure** that is commonly used in databases and file systems for efficient **searching, insertion, deletion**, and **range queries**. It is an extension of the **B-Tree** and is designed to store data in a way that allows quick retrieval, even with large datasets.

Characteristics of a B+ Tree:

1. **Balanced Tree**:
   - All leaves are at the **same level**, ensuring that the time to search, insert, or delete is consistent and logarithmic in nature.

  - This balance guarantees **O(log n)** time complexity for operations.
2. **Internal Nodes**:
   - Internal nodes store only **keys**, not actual data (values). These keys act as **pointers** to direct the search to the appropriate leaf node.
3. **Leaf Nodes**:
   - Leaf nodes store **actual data** or references to the data.
   - They are **linked** in a **linked list**, which enables efficient **range queries** (traversing consecutive leaf nodes).
4. **Ordered Keys**:
   - All keys in a node are stored in a sorted manner, ensuring that search operations follow an ordered sequence.
5. **M-way Tree**:
   - A B+ Tree is an **M-way tree**, meaning each node can have **M** children, where **M** is the order of the tree. The order **M** determines the maximum number of children each node can have.
6. **Search Efficiency**:
   - The **search operation** is similar to a binary search, but because all data is in the leaf nodes, it is efficient for both **single lookups** and **range queries**.

Operations Supported by B+ Tree:

1. **Search**:
   - Search operations start at the **root** node and traverse down the tree to the appropriate leaf node by following pointers. The search complexity is **O(log n)**, where **n** is the number of keys in the tree.
   - **Example**: To find a key, the algorithm follows the pointers from the root through the internal nodes to the leaf node, where the key resides.
2. **Insertion**:
   - Insertion starts by finding the correct leaf node to insert the key. If the leaf node is full, it splits into two nodes, and the middle key is propagated upwards into the parent node.
   - **Example**: If inserting key 15 into a node, it finds the correct leaf and inserts it. If the leaf node is full, it splits, and the middle value is moved up.
3. **Deletion**:
   - Deletion is done by removing the key from the appropriate leaf node. If the deletion causes the leaf node to underflow (i.e., have fewer than the minimum required keys), it may borrow a key from a sibling or merge with a sibling.

   - **Example**: If key 15 is deleted from the tree, it may be merged with a sibling if the leaf node is underfilled.
4. **Range Queries**:
   - The **linked list** of leaf nodes makes range queries particularly efficient. You can quickly traverse through consecutive keys from one leaf node to another to answer range-based queries.
   - **Example**: For a query requesting all values between 15 and 25, the B+ Tree allows you to find 15 and then move through consecutive leaf nodes to retrieve all values up to 25.

Example of a B+ Tree:

Let's consider a **B+ Tree of order 3** (i.e., each node can have a maximum of 3 children):

- **Initial Tree**: Inserting values 10, 20, 5, 6, 8, 15, 25, 30.

Step 1: Insert 10, 20, and 5. The tree looks like this:

```
              [5, 10, 20]
```

Step 2: Insert 6, 8. The tree will split, and the middle key (6) will propagate to the parent node:

```
     [10]
    /   \
 [5, 6]    [8, 20]
```

Step 3: Insert 15, 25, 30. The tree expands, and keys propagate upwards:

```
        [10, 20]
       /      |       \
 [5, 6]   [8]   [15]   [25, 30]
```

Final Tree Structure:

- The **leaf nodes** store the actual data values, and the internal nodes store the keys used for searching.

Advantages of B+ Tree:

1. **Efficient Range Queries**:
   - The linked list of leaf nodes makes it very efficient to perform **range queries**. You can easily traverse all leaf nodes without additional searching.
2. **Balanced Structure**:
   - The B+ Tree remains balanced, providing predictable and consistent performance for insert, delete, and search operations.
3. **Disk Efficiency**:
   - Since B+ Trees store keys in internal nodes and actual data in leaves, they are more disk-efficient, minimizing disk accesses and maximizing the utilization of disk block space.

Disadvantages of B+ Tree:

1. **Complexity**:
   - B+ Trees require complex algorithms for insertion and deletion, especially when handling node splits and merges.

2. **Memory Overhead**:

- Due to the additional linked list in the leaves, the B+ Tree may require more memory compared to other data structures like B-Trees.