R18

Code No: 154AM

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B. Tech II Year II Semester Examinations, April/May - 2023
DATABASE MANAGEMENT SYSTEMS
(Common to CSE, IT, ECM, CSBS, CSIT, ITE, CSE(AI&ML), CSE(DS))

Time: 3 Hours **Max. Marks: 75**

Note: i) Question paper consists of Part A, Part B.
ii) Part A is compulsory, which carries 25 marks. In Part A, Answer all questions.
iii) In Part B, Answer any one question from each unit. Each question carries 10 marks and may have a, b as sub questions.

PART – A

(25 Marks)

1.a) What are the goals of DBMS? [2]
b) Explain about DML language and query processor. [3]
c) Distinguish between super key and Candidate key. [2]
d) Explain Domain relational calculus. [3]
e) Define dependency preserving decomposition. [2]
f) What is the difference between 3NF and BCNF? [3]
g) Explain about durability of transaction. [2]
h) What is transaction? Explain its states. [3]
i) Why are tree-structure indexes are good for searches, especially range selections. [2]
j) What is the main difference between ISAM and B+ tree indexes? [3]

PART – B

(50 Marks)

2.a) Identify the main components in a DBMS and briefly explain what they do?
b) Explain the following:
i) View of Data
ii) Data Abstraction
iii) Instances and Schemas. [5+5]

OR

3.a) What is data model? Explain Relational Model and E-R model.
b) Draw an ER-Diagram for Library Management system. [5+5]

4.a) Differentiate between a relation schema and relation instance define the term arity and degree of a relation.
b) Let R =(ABC) and let r1 and r2 both relations on schema R . Give an expression in the Domain relational calculus that is equivalent to each of the following: [5+5]
i) $\prod A(r1)$ ii) $\sigma\ B=17(r1)$ iii) $r1 \cap r2$

OR

5.a) What is Relational Model? Explain about various domain and integrity constraints in Relational Model with examples.
b) Explain various fundamental operations in relational algebra with examples. [5+5]

6.a) What aggregate operators does SQL support ? Explain.
b) Define Functional dependencies and Multi valued dependencies. How are primary keys related to FDs? [5+5]

OR

7.a) What are the conditions are required for a relation to be in 4NF and 3NF explain with examples.
b) Explain various set operations are used in SQL with examples. [5+5]

8.a) What is locking Protocol? Describe the Strict Two Phase locking Protocol.
b) Explain multiple granularity concurrency control scheme. [5+5]

OR

9.a) Explain the ACID Properties of transactions.
b) What is log file? Explain the following log based recovery schemes.
i) Deferred data base modification
ii) Immediate data base modification. [5+5]

10.a) Explain about cluster index, primary and secondary indexes with examples.
b) Explain Deletion and insertion operations in ISAM with examples. [5+5]

OR

11.a) Explain what are the differences between tree based and Hash based indexes.
b) Explain deletion and insertion operation in *B+ trees*. [4+6]

---ooOoo---

ANSWER KEY

PART – A

1.

a) What are the goals of DBMS?

Sol: The primary goals of a Database Management System (DBMS) are to store data efficiently, allow for easy retrieval and manipulation of data, ensure data integrity and security, support data concurrency, and provide a mechanism for data recovery in case of failures.

b) Explain about DML language and query processor.

Sol: DML is used for accessing and manipulating data in a database, including operations like SELECT, INSERT, UPDATE, and DELETE. The query processor interprets and executes DML statements, optimizing query execution and managing the interaction with the database.

c) Distinguish between super key and Candidate key.

Sol: A super key is any set of attributes that uniquely identifies a record in a table. A candidate key is a minimal super key, meaning it has no unnecessary attributes; it uniquely identifies a record and no subset of it can do the same.

d) Explain Domain relational calculus.

Sol: Domain Relational Calculus (DRC) is a type of query language for relational databases where queries are expressed as formulas consisting of variables that range over domain elements, using logical connectives and quantifiers to specify the retrieval of data.

e) Define Dependency Preserving Decomposition.

Sol: This is a property of database decomposition where the original set of functional dependencies is preserved after decomposing the database schema into two or more schemas. This ensures that constraints are not lost and can still be enforced.

f) What is the difference between 3NF and BCNF?

Sol: Third Normal Form (3NF) ensures that all attributes are functionally dependent only on the primary key, while Boyce-Codd Normal Form (BCNF) is a stricter version where every determinant is a candidate key, eliminating any potential redundancy.

g) Explain about durability of transaction.

Sol: Durability ensures that once a transaction has been committed, its effects are permanently recorded in the database, even in the event of a system failure. This property is crucial for maintaining data integrity.

h) What is transaction? Explain its states.

Sol: A transaction is a sequence of database operations that are treated as a single unit of work. Its states include:

- **Active**: The transaction is executing.
- **Partially Committed**: The final operation is being executed.
- **Committed**: All operations are completed successfully.
- **Failed**: An error has occurred, causing the transaction to be rolled back.
- **Aborted**: The transaction has been rolled back and all changes undone.

i) Why are tree-structure indexes are good for searches, especially range selections

Sol: Tree-structure indexes, like B-trees, are effective for searches and range selections because they provide a balanced structure that maintains sorted order, allowing for efficient insertion, deletion, and retrieval operations with logarithmic time complexity.

j) What is the main difference between ISAM and B+ tree indexes?

Sol: ISAM (Indexed Sequential Access Method) is a static indexing method where the index structure is fixed after creation, leading to potential inefficiency over time. B+ Trees, on the other hand, are dynamic, allowing for efficient updates and maintaining balanced structure for optimal search performance.

PART – B

2.

a) Identify the main components in a DBMS and briefly explain what they do?

Sol:

The main components in a DBMS and their functions are:

1. **Database Engine**:
    - **Function**: Handles data storage, retrieval, and update operations. It is responsible for the actual interaction with the database files on disk.

2. **Query Processor**:
    - **Function**: Interprets and executes database queries. It breaks down DML (Data Manipulation Language) queries into low-level instructions the database engine can understand and optimizes their execution for efficiency.
3. **Transaction Manager**:
    - **Function**: Ensures that database transactions are processed reliably and adhere to the ACID properties (Atomicity, Consistency, Isolation, Durability). It manages transaction states, concurrency control, and recovery from failures.
4. **Storage Manager**:
    - **Function**: Manages how data is stored on disk, including data structures, file organization, and access methods. It ensures efficient storage, retrieval, and updating of data.
5. **Metadata Catalog** (or **Data Dictionary**):
    - **Function**: Contains metadata, which is data about the data. This includes data definitions, schema information, table structures, indexes, constraints, and relationships. It helps the DBMS understand the structure and organization of the database.

b) Explain the following:
i) View of Data
ii) Data Abstraction
iii) Instances and Schemas.

Sol:

i) View of Data: In a DBMS, data can be viewed at three levels:

Physical Level: How data is actually stored on the hardware. -**Logical Level**: The structure of the entire database as understood by the DBMS, using tables and relationships. - **View Level**: Specific views of the database tailored to user needs, showing only relevant data and hiding the rest.

ii) Data Abstraction: The process of hiding the complexities of the database's physical storage and presenting users with a simpler interface.

This includes: - **Physical Abstraction**: Details about how data is stored. - **Logical Abstraction**: Structure and organization of data. - **View Abstraction**: User-specific views of data.

iii) Instances and Schemas:

Instance: The actual content of the database at a particular moment in time; the data stored in the database. - **Schema**: The overall design or structure of the database, defined at creation and usually not frequently changed. It describes the tables, relationships, and constraints in the database.

OR

3.

a) What is data model? Explain Relational Model and E-R model.

Sol: Data Model: A data model is a conceptual framework for organizing and defining the structure of a database. It dictates how data is stored, accessed, and manipulated within the database system, including the relationships among data elements and the rules governing data integrity.

Relational Model:

- **Overview**: The relational model organizes data into tables (relations) consisting of rows and columns. Each table represents an entity, and each row represents a record, while columns represent the attributes of the entity.
- **Components**:
  - **Tables (Relations)**: Each table has a unique name and consists of rows (tuples) and columns (attributes).
  - **Attributes**: Columns in a table, each representing a property of the entity.
  - **Tuples**: Rows in a table, each representing a single record.
  - **Keys**: Unique identifiers for rows within a table, including primary keys and foreign keys.
  - **Integrity Constraints**: Rules to maintain data accuracy and consistency, such as entity integrity (each table must have a primary key) and referential integrity (foreign keys must match primary keys in related tables).
- **Example**: A table named "Students" with columns "StudentID", "Name", and "Major". "StudentID" could be the primary key.

Entity-Relationship (E-R) Model:

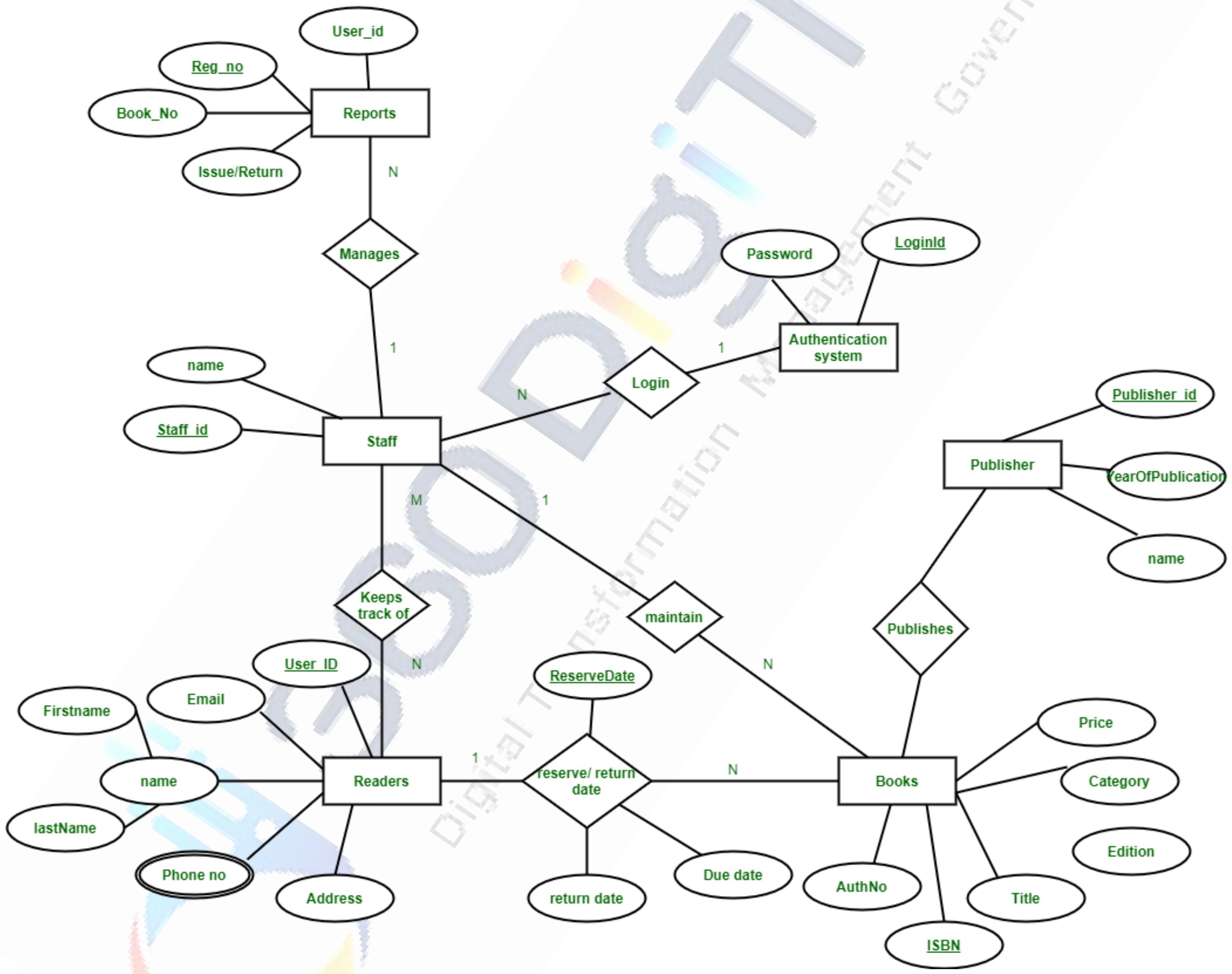
- **Overview**: The E-R model is a high-level conceptual data model that defines data elements and their relationships. It uses entities, attributes, and relationships to represent data.
- **Components**:
  - **Entities**: Objects or concepts that can have data stored about them. Entities are represented by rectangles.
  - **Attributes**: Properties or characteristics of entities. Attributes are represented by ovals.
  - **Relationships**: Associations between entities. Relationships are represented by diamonds.
  - **Entity Sets**: Collections of similar entities.
  - **Relationship Sets**: Collections of similar relationships.
  - **Primary Keys**: Unique identifiers for entities.

- **Cardinality**: Defines the number of instances of one entity that can be associated with instances of another entity (one-to-one, one-to-many, many-to-many).

- **Example**: An E-R diagram representing students and courses. Entities "Student" and "Course" with a relationship "Enrolls". "Student" might have attributes "StudentID", "Name", and "Major", while "Course" might have attributes "CourseID" and "Title". The "Enrolls" relationship links "Student" and "Course" entities.

b) Draw an ER-Diagram for Library Management system.

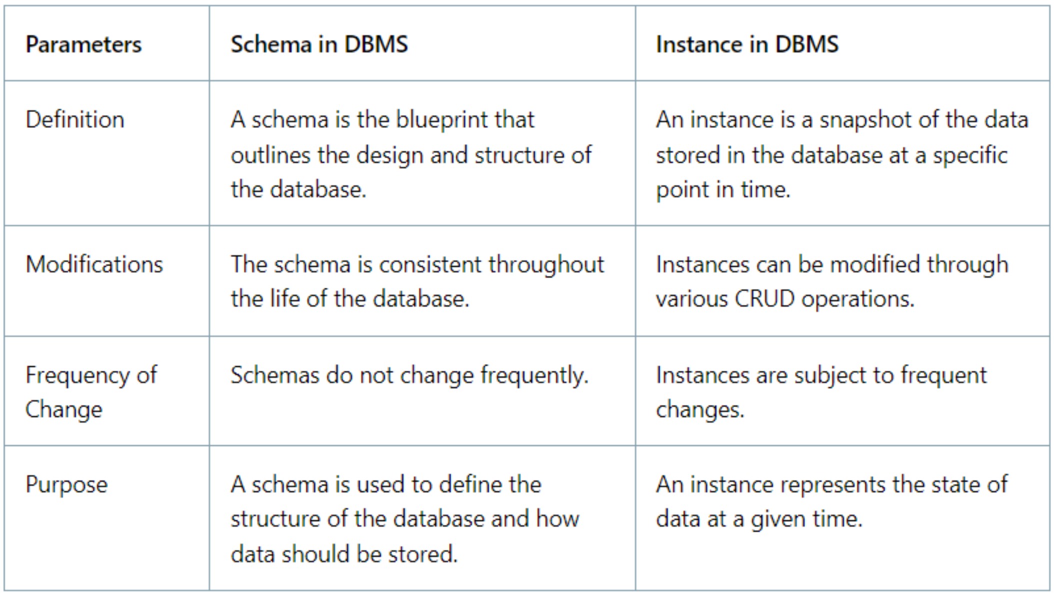
Sol:



4.

a) Differentiate between a relation schema and relation instance define the term arity and degree of a relation.

Sol:



Parameters	Schema in DBMS	Instance in DBMS
Definition	A schema is the blueprint that outlines the design and structure of the database.	An instance is a snapshot of the data stored in the database at a specific point in time.
Modifications	The schema is consistent throughout the life of the database.	Instances can be modified through various CRUD operations.
Frequency of Change	Schemas do not change frequently.	Instances are subject to frequent changes.
Purpose	A schema is used to define the structure of the database and how data should be stored.	An instance represents the state of data at a given time.

Arity and Degree of a Relation:

Arity (Degree):

- **Definition**: Arity, also known as the degree of a relation, refers to the number of attributes (columns) in a relation schema.
- **Example**: In the "Students" relation schema Students(StudentID, Name, Major), the arity (degree) is 3 because there are three attributes: StudentID, Name, and Major.

b) Let R =(ABC) and let r1 and r2 both relations on schema R . Give an expression in the Domain relational calculus that is equivalent to each of the following:

i) ∏A(r1)   ii) σ B=*17*(r1)   iii) r1∩r2

Sol:

i) Projection (∏A(r1))

Projection selects specific attributes from a relation. In DRC, we express projection using existential quantifiers (∃) and conjunctions (∧).

Expression: {t|∃B,C(r1(A,B,C)∧t(A))}

Explanation: This expression selects all tuples ttt where there exist attributes B and C such that r1(A,B,C)r1(A, B, C)r1(A,B,C) holds, and ttt includes only attribute A.

ii) Selection (σ B=17(r1))

Selection filters rows that satisfy a specific condition (predicate). In DRC, selection is expressed using existential quantifiers and the condition itself.

Expression: {t|∃A,C(r1(A,17,C)∧t(A,17,C))}

Explanation: This expression selects all tuples *t* where there exist attributes A and C such that r1(A,17,C)r1(A, 17, C)r1(A,17,C) holds, and *t* includes these attributes.

iii) Intersection (r1 ∩ r2)

Intersection combines tuples that are common between two relations. In DRC, we express intersection using conjunctions and existentially quantified variables.

Expression: {t|∃A,B,C((r1(A,B,C)∧r2(A,B,C))∧t(A,B,C))}

Explanation: This expression selects all tuples *t* where there exist attributes A, B and C such that both r1(A,B,C)r1(A, B, C)r1(A,B,C) and r2(A,B,C)r2(A, B, C)r2(A,B,C) hold, and *t* includes these attributes.

OR

5.
a) What is Relational Model? Explain about various domain and integrity constraints in Relational Model with examples.

Sol: The **Relational Model** is a conceptual framework for organizing and managing data in a database. It represents data as tables (relations) consisting of rows (tuples) and columns (attributes), where each table represents an entity type and each row represents a unique record.

Various Domain and Integrity Constraints in Relational Model:

1. Domain Constraints:

- **Definition**: Domain constraints define the permissible values that can be stored in a column (attribute) of a table. It specifies the data type, format, and range of values that a particular attribute can hold.

Examples of Domain Constraints:

- **Data Types**: Specifies the type of data allowed in an attribute (e.g., integer, string, date).

- Example: In a Students table, the StudentID attribute might be constrained to hold only integers.
- **Length Constraints**: Limits the length of string data that can be stored in an attribute.
  - Example: The Name attribute in the Students table might be limited to 50 characters.
- **Range Constraints**: Specifies allowable numerical values for an attribute.
  - Example: The Age attribute in a Employees table might be constrained to values between 18 and 65.
- **Nullability**: Specifies whether an attribute can be NULL (missing or undefined).
  - Example: The DateOfBirth attribute in a Customers table might allow NULL values if the birthdate is unknown.

2. Integrity Constraints:

- **Definition**: Integrity constraints ensure the accuracy, consistency, and reliability of data stored in the database. They enforce rules that govern relationships between tables and maintain the correctness of data during insertions, updates, and deletions.

Examples of Integrity Constraints:

- **Entity Integrity Constraint**: Ensures that each row (tuple) in a table has a unique identifier (primary key).
  - Example: In a Students table, the StudentID attribute is designated as the primary key, ensuring each student record has a unique identifier.
- **Referential Integrity Constraint**: Ensures that relationships between tables are maintained correctly. It typically involves foreign keys that reference primary keys in related tables.
  - Example: In a Courses table, the InstructorID attribute might reference the EmployeeID attribute in an Instructors table, ensuring that only valid instructor IDs can be assigned to courses.
- **Domain Constraints**: Ensure that data values are within defined domains.
  - Example: A Gender attribute in a Employees table might be constrained to only allow values 'Male' or 'Female'.
- **Check Constraints**: Specifies conditions that must be true for data to be valid.
  - Example: A Salary attribute in an Employees table might have a check constraint that ensures the salary is greater than or equal to zero.

b) Explain various fundamental operations in relational algebra with examples

Sol:
Fundamental Operations in Relational Algebra

Relational algebra is a procedural query language that works on the concept of mathematical set operations applied to relations (tables). Here are the fundamental operations:

1. **Selection (σ)**
   - **Definition**: Selects rows from a relation that satisfy a given predicate.
   - **Example**: Select students with age greater than 18.

     Sql: SELECT * FROM Student WHERE Age > 18;

2. **Projection (π)**
   - **Definition**: Selects specific columns from a relation.
   - **Example**: Select the names and ages of students.

     Sql: SELECT Name, Age FROM Student;

3. **Union (∪)**
   - **Definition**: Combines the results of two relations, eliminating duplicate rows.
   - **Example**: Combine two sets of students.

     Sql: SELECT * FROM Student1 UNION SELECT * FROM Student2;

4. **Set Difference (−)**
   - **Definition**: Returns the rows that are in the first relation but not in the second.
   - **Example**: Find students enrolled in Course1 but not in Course2.

     Sql: SELECT * FROM Course1_Students EXCEPT SELECT * FROM Course2_Students;

5. **Cartesian Product (×)**
   - **Definition**: Returns the Cartesian product of two relations, combining every row of the first relation with every row of the second relation.
   - **Example**: Combine students and courses to get all possible student-course pairs.

     Sql: SELECT * FROM Student, Course;

6. **Rename (ρ)**
   - **Definition**: Renames the output relation or attributes.
   - **Example**: Rename the attributes of the student relation.

sql
ρ (NewStudentName(Name, Age)) (Student)

7. **Natural Join (⋈)**
   - **Definition**: Combines two relations based on their common attributes.
   - **Example**: Join students and enrollments to get student-course pairs.

     Sql: SELECT * FROM Student NATURAL JOIN Enrollment;

8. **Division (÷)**
   - **Definition**: Returns those tuples in one relation that are associated with all tuples in another relation.
   - **Example**: Find students who are enrolled in all courses offered.

     sql
     Students ÷ Courses

6.

a) What aggregate operators does SQL support? Explain.

Sol: SQL (Structured Query Language) supports several aggregate operators that allow for the calculation and summarization of data within a query result set. These operators perform operations across multiple rows of a table and return a single result. Here are the main aggregate operators supported by SQL:

1. COUNT():

- **Function**: Counts the number of rows that match a specified condition.
- **Example**: Count the number of students in the Students table.

```
SELECT COUNT(*) FROM Students;
```

2. SUM():

- **Function**: Calculates the sum of values in a numeric column.
- **Example**: Calculate the total salary paid to all employees in the Employees table.

```
SELECT SUM(Salary) FROM Employees;
```

3. AVG():

- **Function**: Calculates the average of values in a numeric column.
- **Example**: Calculate the average GPA of students in the Students table.

```
SELECT AVG(GPA) FROM Students;
```

4. MIN():

- **Function**: Finds the minimum value in a column.
- **Example**: Find the minimum age of employees in the Employees table.

```
SELECT MIN(Age) FROM Employees;
```

5. MAX():

- **Function**: Finds the maximum value in a column.
- **Example**: Find the maximum salary of employees in the Employees table.

```
SELECT MAX(Salary) FROM Employees;
```

6. GROUP BY:

- **Clause**: Groups rows that have the same values into summary rows, like "find the number of customers in each country".
- **Example**: Group employees by department and calculate the total salary for each department.

```
SELECT Department, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Department;
```

7. HAVING:

- **Clause**: Filters records that satisfy a specified condition after a GROUP BY clause has been applied.
- **Example**: Find departments with a total salary greater than $100,000.

```
SELECT Department, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Department
HAVING SUM(Salary) > 100000;
```

b) Define Functional dependencies and Multi valued dependencies. How are primary keys related to FDs?

Sol:

1. Functional Dependencies (FDs):

A functional dependency is a constraint between two sets of attributes in a relation from a database. It specifies that the values of one set of attributes determine the values of another set. Formally, if we have attributes X and Y in a relation R, X determines Y (denoted as $X \rightarrow Y$) if and only if for every instance of X, there is exactly one instance of Y.

2. Multi-Valued Dependencies (MVDs):

A multi-valued dependency is a constraint between three sets of attributes in a relation. It specifies that if two tuples agree on a certain set of attributes (say X), then certain other attributes (say Y) must have related values. Formally, if in a relation R, for attributes X, Y, Y, Z, an MVD X↠Y specifies that for every combination of values of X, there is a corresponding set of values of Y, independent of Z.

Relationship to Primary Keys:

- **Primary Keys (PKs):**
    - A primary key is a set of attributes in a relation that uniquely identifies each tuple (row) in that relation. It uniquely identifies a record within a table.
- **Relationship:**
    - Primary keys are directly related to functional dependencies because they ensure that each attribute or combination of attributes (in the case of composite keys) uniquely identifies a row. FDs ensure that certain attributes functionally determine others, which is crucial for maintaining data integrity and avoiding redundancies.
    - Multi-valued dependencies, on the other hand, are concerned with the relationships between attributes rather than uniqueness, ensuring that certain sets of attributes have specific relationships when other sets are held constant.

OR

7.
a) What are the conditions are required for a relation to be in 4NF and 3NF explain with examples.

Sol:
Third Normal Form (3NF)

A relation is in Third Normal Form (3NF) if it meets the following conditions:

1. **It is in Second Normal Form (2NF):** This means the relation is in First Normal Form (1NF) and all non-prime attributes are fully functionally dependent on the primary key.
2. **No transitive dependency:** A non-prime attribute should not depend on another non-prime attribute. Formally, for a functional dependency X→Y, either X is a superkey, or Y is a prime attribute (an attribute that is part of a candidate key).

Example:

Consider a relation Employee with attributes EmployeeID, EmployeeName, DepartmentID, and DepartmentName:

- EmployeeID→EmployeeName
- DepartmentID→DepartmentName

If DepartmentID is not part of the primary key:

- There is a transitive dependency: EmployeeID→DepartmentID→DepartmentName
- This violates 3NF.

To convert it to 3NF:

1. Decompose the relation into two relations:
   - Employee(EmployeeID, EmployeeName, DepartmentID)
   - Department(DepartmentID, DepartmentName)
2. Now, each non-prime attribute is fully dependent on a candidate key, and there are no transitive dependencies.

Fourth Normal Form (4NF)

A relation is in Fourth Normal Form (4NF) if it meets the following conditions:

1. **It is in Boyce-Codd Normal Form (BCNF):** This means the relation is in 3NF, and for every functional dependency X→ Y, X is a superkey.
2. **No multi-valued dependencies (MVDs):** A relation should not have more than one multi-valued dependency. A multi-valued dependency X→→Y occurs if, for each value of X, there is a set of values for Y independent of other attributes.

Example:

Consider a relation Course with attributes CourseID, Instructor, and Textbook:

- A course can be taught by multiple instructors.
- A course can use multiple textbooks.

This results in a multi-valued dependency:

- CourseID→→Instructor
- CourseID→→Textbook

To convert it to 4NF:

1. Decompose the relation into two relations:
   - CourseInstructor(CourseID, Instructor)
   - CourseTextbook(CourseID, Textbook)

2. Now, each relation has only one multi-valued dependency, satisfying 4NF.

b) Explain various set operations are used in SQL with examples.

Sol: Set Operations

Set operations in the context of databases involve manipulating the result sets of queries to combine, compare, or extract data from multiple tables or queries. The main set operations typically supported by SQL are UNION, INTERSECT, and EXCEPT (or MINUS in some databases).

Example Queries:

UNION: Combines the result sets of two or more SELECT statements into a single result set, removing duplicate rows.

Example: Suppose we have two tables Employees and Contractors with similar structures:

Employees table

SELECT EmployeeID, Name, 'Employee' as Type FROM Employees UNION

Contractors table

SELECT ContractorID, Name, 'Contractor' as Type FROM Contractors;

This query combines employee and contractor data into a single result set, adding a Type column to differentiate between them.

INTERSECT: Returns only the rows that appear in both result sets of two queries.

Example: Finding common employees between two departments:

```
SELECT EmployeeID, Name
FROM Employees
WHERE Department = 'IT'
INTERSECT
SELECT EmployeeID, Name
FROM Employees
WHERE Department = 'HR';
```

This query retrieves employees who belong to both the IT and HR departments.

EXCEPT (or MINUS): Returns the rows that appear in the first result set but not in the second result set.

Example: Finding employees who are not contractors:

```
SELECT EmployeeID, Name
FROM Employees
EXCEPT
SELECT EmployeeID, Name
FROM Contractors;
```

This query retrieves employees who are not also listed as contractors.

8.

a) What is locking Protocol? Describe the Strict Two Phase locking Protocol.

Sol:

Locking Protocol

A locking protocol is a set of rules used to ensure that transactions in a database management system (DBMS) are executed in a safe and consistent manner. It manages the access to database objects (such as rows or tables) to prevent concurrent transactions from interfering with each other, which could lead to data inconsistency or corruption.

Locking protocols typically involve:

1. **Locks:** Mechanisms to control access to data items.
   - **Shared Lock (S-lock):** Allows a transaction to read a data item. Multiple transactions can hold shared locks on the same item.
   - **Exclusive Lock (X-lock):** Allows a transaction to read and write a data item. Only one transaction can hold an exclusive lock on an item at a time.
2. **Locking Rules:** Rules that transactions must follow to acquire and release locks.

Strict Two-Phase Locking (Strict 2PL)

Strict Two-Phase Locking (Strict 2PL) is a popular and widely used locking protocol that ensures serializability (the correctness of transactions) by enforcing two key phases for transaction execution:

1. **Growing Phase:**

- A transaction may acquire locks (both shared and exclusive) but cannot release any locks.

2. **Shrinking Phase:**
    - A transaction releases all its locks. Once a transaction releases a lock, it cannot acquire any new locks.

Strict 2PL Extension:

- All locks held by a transaction are released only when the transaction commits or aborts. This ensures that no other transactions can access the data being modified by the current transaction until it is completely done (committed or aborted).

b) Explain multiple granularity concurrency control scheme.

Sol: A multiple granularity concurrency control scheme (MGCC) is a technique used in database management systems (DBMS) to manage concurrent access to data at different levels of granularity, such as at the level of individual data items (like rows or columns) or larger units (like entire tables or partitions). This scheme is crucial for ensuring efficient and correct concurrent access to data by multiple transactions.

Key Concepts in MGCC:

1. **Granularity Levels:**
    - **Fine-granularity:** Involves smaller units of data, such as individual rows or even columns within a table.
    - **Coarse-granularity:** Involves larger units of data, such as entire tables or partitions.
2. **Concurrency Control Mechanisms:**
    - **Locking:** MGCC typically employs locking mechanisms to control concurrent access to data. Locks can be applied at different levels of granularity:
        - **Table-level locks:** Lock entire tables.
        - **Page-level locks:** Lock pages of data (groups of contiguous data blocks).
        - **Row-level locks:** Lock individual rows or even columns within rows.
    - **Multiversioning:** Some MGCC schemes use multiversion concurrency control (MVCC), where multiple versions of data items are maintained to allow for concurrent reads and writes without blocking.
3. **Hierarchical Locking:**
    - MGCC often uses a hierarchical approach where locks at finer levels of granularity are compatible with locks at coarser levels but not vice versa. For example, a row-level lock should be compatible with a table-level lock, but a table-level lock might conflict with a row-level lock.

4. **Deadlock Avoidance:**
   - MGCC schemes must include mechanisms to detect and resolve deadlocks, which occur when transactions wait indefinitely for resources locked by each other.

OR

9.
a) Explain the ACID Properties of transactions.

Sol: The ACID properties are a set of principles that ensure reliable processing of database transactions. They guarantee that transactions are processed in a secure, consistent, and predictable manner, which is crucial for maintaining the integrity of the database. The acronym ACID stands for Atomicity, Consistency, Isolation, and Durability.

Atomicity:

Definition: A transaction is an indivisible unit of work that is either fully completed or fully rolled back. This means that a transaction must be all-or-nothing: if any part of the transaction fails, the entire transaction fails and the database state is left unchanged.

Example: Consider a banking system where you transfer money from Account A to Account B. The transaction involves debiting Account A and crediting Account B. Atomicity ensures that either both operations are completed successfully, or neither operation is performed, so that the accounts remain consistent.

Consistency:

Definition: A transaction must transform the database from one valid state to another valid state. This means that a transaction should leave the database in a consistent state, adhering to all defined rules, constraints, and triggers.

Example: In the same banking system, a rule might be that the total balance of all accounts must remain constant. If the transfer from Account A to Account B violates this rule, the transaction would be rolled back, ensuring consistency.

Isolation:

Definition: Transactions should be executed in isolation from one another. The operations of one transaction should not be visible to other transactions until the transaction is committed. This prevents transactions from interfering with each other.

Example: Suppose two transactions are running simultaneously: one debits Account A and the other credits Account B. Isolation ensures that the intermediate state of these operations is not visible to other transactions, preventing inconsistent views of the database.

Durability:

Definition: Once a transaction has been committed, its changes to the database are permanent, even in the event of a system failure. This means that committed data is saved reliably and must be recoverable.

Example: After the money transfer transaction is completed and committed, the changes (debiting Account A and crediting Account B) are permanent. Even if the system crashes immediately after the commit, the changes are preserved and can be recovered when the system restarts.

b) What is log file? Explain the following log based recovery schemes.

i) **Deferred data base modification**

ii) **Immediate data base modification.**

Sol:

Log File: A log file is a crucial component of database management systems (DBMS) used for recovery purposes. It is a sequential file that records all the changes made to the database, ensuring that transactions can be tracked and, if necessary, rolled back or replayed to maintain data consistency. The log file typically contains information about each transaction, such as:

- Transaction ID
- Type of operation (insert, update, delete)
- Affected data items
- Old and new values of the data items
- Timestamps
- Commit or abort status

By maintaining a detailed log, the DBMS can recover from crashes and ensure that the database remains consistent and durable.

1. Deferred Database Modification

Concept: In the deferred database modification scheme, updates to the database are not applied until the transaction commits. This means that changes are recorded in the log file but not immediately written to the database.

Process:

1. **Transaction Start:** When a transaction begins, a log entry <T, start> is written.
2. **Operations:** All changes (insert, update, delete) are logged as <T, X, new_value>, but the database is not updated.
3. **Commit:** When the transaction commits, a <T, commit> entry is written, and the deferred changes are applied to the database.
4. **Recovery:** In case of a crash, only committed transactions' changes are applied. Uncommitted transactions are ignored.

2. Immediate Database Modification

Concept: In the immediate database modification scheme, updates are applied to the database as soon as they are performed. Changes are also logged before being written to the database, ensuring that they can be undone if necessary.

Process:

1. **Transaction Start:** When a transaction begins, a log entry <T, start> is written.
2. **Operations:** Each change is logged as <T, X, old_value, new_value> before the database is updated.
3. **Commit:** When the transaction commits, a <T, commit> entry is written.
4. **Recovery:** In case of a crash, the log is used to undo uncommitted transactions and redo committed transactions.

10.

a) Explain about cluster index, primary and secondary indexes with examples.
Sol:

1) **Cluster Index:**

Definition: A cluster index determines the physical order of data in a table. The rows are stored in the same order as the index. This type of index is particularly useful for range queries, as it allows for faster retrieval of contiguous data blocks.

Example: Consider a table Students with columns StudentID, Name, and DOB (Date of Birth). If we create a cluster index on DOB, the table rows will be physically ordered by the DOB column.

SQL Example:

```
CREATE CLUSTERED INDEX idx_students_dob ON Students(DOB);
```

2) Primary Index

Definition: A primary index is an index on the primary key of a table. It is usually unique and ensures that no two rows have the same value in the indexed column(s). The primary index can be clustered or non-clustered.

Example: Consider a table Employees with EmployeeID as the primary key. Creating a primary index on EmployeeID ensures that each EmployeeID is unique and serves as a quick lookup for any query involving EmployeeID.

SQL Example:

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(100),
   Position VARCHAR(50)
);
```

3) Secondary Index

Definition: A secondary index is an index that is not the primary key. It provides a mechanism to access data based on non-primary key columns. Secondary indexes can be used to improve the performance of queries that filter or sort on columns other than the primary key.

Example: Consider the same Employees table. If we frequently query the table based on the Position column, we can create a secondary index on Position to speed up these queries.

SQL Example:

```
CREATE INDEX idx_employees_position ON Employees(Position);
```

b) Explain Deletion and insertion operations in ISAM with examples.

Sol:

1) Insertion Operation

Process:

1. **Locate the Block:** Use the primary index to find the appropriate block for the new record.
2. **Check Space:**
   - If there is space in the block, insert the record directly.
   - If the block is full, insert the record into the overflow area linked to the block.

Example: Suppose we have a Students table with StudentID as the primary index. To insert StudentID 105:

- Locate the block for IDs 100-109.
- If the block has space, insert 105 directly.
- If the block is full, add 105 to the overflow area.

2) Deletion Operation

Process:

1. **Locate the Record:** Use the primary index to find the record.
2. **Mark as Deleted:** Mark the record as deleted or remove it from the overflow area if applicable.

Example: To delete StudentID 103 from the Students table:

- Locate the block containing 103.
- Mark 103 as deleted.

OR

11.

a) Explain what are the differences between tree based and Hash based indexes.

Sol:

	Tree-Based Indexes	**Hash-Based Indexes**
Structure	Tree-Based Indexes are hierarchical, balanced, sorted in structure	Hash-Based Indexes are hash table, buckets in structure
Best For	Range queries, ordered traversals.	Equality searches.
Example Use Cases	B-trees, B+-trees.	Hash indexes in databases, hash maps.
Advantages	Efficient for range queries, dynamic handling of inserts/deletes.	Fast point queries, simpler maintenance.
Disadvantages	Complex maintenance, potentially slower point queries.	Inefficient for range queries, collision handling

b) Explain deletion and insertion operation in *B+ trees*.

Sol:

Insertion Operation in B+ Trees

1. **Locate the Leaf Node:** Traverse the tree from the root to find the appropriate leaf node for the new key.
2. **Insert the Key:**
   - If the leaf node has space, insert the key in sorted order.
   - If the leaf node is full, split the node into two and promote the middle key to the parent node.
3. **Handle Splits:** If the parent node becomes full due to the promotion, recursively split and promote keys up the tree.
4. **Update Tree Structure:** Ensure that all pointers and references are updated to maintain the B+ tree properties.

Deletion Operation in B+ Trees

1. **Locate the Leaf Node:** Traverse the tree from the root to find the leaf node containing the key to be deleted.
2. **Delete the Key:** Remove the key from the leaf node.
3. **Handle Underflow:**
   - If the leaf node has fewer keys than the minimum required, either borrow a key from a sibling or merge with a sibling.
   - Update the parent node accordingly.
4. **Update Parent Nodes:** If merging or borrowing causes the parent to underflow, handle it recursively up the tree to ensure balance.

---ooOoo---