20CB403/22CM403/20CS403/20DS403/20IT403

Hall Ticket Number:

II/IV B. Tech (Regular) DEGREE EXAMINATION

July, 2025 — **Common to CB, CM, CS, DS & IT**

Fourth Semester — **Database Management System**

Time: Three Hours — **Maximum:** 70 Marks

Answer question 1 compulsorily. **(14X1 = 14 Marks)**

Answer one question from each unit. **(4X14 = 56 Marks)**

			CO	BL	M
1	a)	What is the role of database administrator?	CO 1	L1	1M
	b)	Define data independence.	CO 1	L1	1M
	c)	Define centralized database system.	CO 1	L1	1M
	d)	Define entity set.	CO 1	L1	1M
	e)	Differentiate relational algebra and relational calculus.	CO 2	L4	1M
	f)	List the different datatypes in SQL.	CO 2	L1	1M
	g)	Why do we use JOIN in SQL?	CO 2	L1	1M
	h)	Define B-Tree.	CO 3	L1	1M
	i)	What is meant by functional dependency.	CO 3	L1	1M
	j)	Define Normal Form.	CO 3	L1	1M
	k)	List various types of failures occur in transaction.	CO 4	L1	1M
	l)	Define ROLLBACK.	CO 4	L1	1M
	m)	What is meant by timestamp?	CO 4	L1	1M
	n)	Define shadow paging.	CO 4	L1	1M
		Unit-I			
2	a)	Explain the detailed system architecture of DBMS with a neat diagram.	CO 1	L2	7M
	b)	Construct an ER diagram for a hospital with a set of patients and a set of medical doctors.	CO 1	L3	7M
		(OR)			
3	a)	Explain Three level schema architecture in detail.	CO 1	L2	7M
	b)	Discuss relationship types, sets, roles, and structural constraints in the context of ER modeling.	CO 1	L2	7M
		Unit-II			
4	a)	Explain about Tuple Relational Calculus (TRC).	CO 2	L3	8M
	b)	Explain SQL aggregate operators with examples.	CO 2	L2	6M
		(OR)			
5	a)	Explain binary relational operations with examples.	CO 2	L2	8M
	b)	Consider following relations and write SQL queries for given statements. Assume suitable constrains. Instructor(ID, Name, Dept_name , Salary) Teaches(ID, Course_id, Sec_id, Semester(even/odd),Year) i) Find the average salary of the instructors in computer department. ii) Find the number of instructors in each department who teach a course in even semester of 2024. iii) Find the names of instructor with salary amounts between 30000 and 50000.	CO 2	L3	6M
		Unit-III			
6	a)	Construct B+ tree of order 3 with the following values 1,2,5,8,10,13,18,21,26,37.	CO 3	L3	7M
	b)	Explain about BCNF and list the differences between BCNF and 3NF.	CO 3	L2	7M
		(OR)			
7	a)	Explain the concepts of Functional dependency with suitable example and its operations like closure.	CO 3	L2	7M
	b)	Consider schema R = (A, B, C, G, H, I) and the set F of functional dependencies {A→ B, A→C, CG→H, CG→I, B→H}. Compute the candidate keys of the schema. Compute the closure of the same.	CO 3	L4	7M
		Unit-IV			
8	a)	Define locking protocol. Explain the Two-Phase Locking protocol with an example.	CO 4	L2	7M
	b)	Differentiate serializable schedule, recoverable schedule and strict schedule.	CO 4	L4	7M
		(OR)			
9	a)	Explain about ACID properties with an example.	CO 4	L2	7M
	b)	Discuss recovery techniques based on immediate update with example.	CO 4	L2	7M

20CB403/22CM403/20CS403/20DS403/20IT403

II/IV B. Tech (Regular) DEGREE EXAMINATION

July, 2025 — **Common to CB, CM, CS, DS & IT**

Fourth Semester — **Database Management System**

Time: Three Hours — **Maximum:** 70 Marks

Answer question 1 compulsorily. **(14X1 = 14 Marks)**

Answer one question from each unit. **(4X14 = 56 Marks)**

			CO	BL	M
1	a)	**What is the role of database administrator?** A database administrator (DBA) is responsible for the design, implementation, maintenance, and security of a database.	CO 1	L1	1M
	b)	**Define data independence.** Data independence is the ability to modify the schema (structure) of a database at one level without requiring changes to the schema at the next higher level.	CO 1	L1	1M
	c)	**Define centralized database system.** A centralized database system is one where all data is stored, located, and maintained in a single, central location, typically on a central computer or server.	CO 1	L1	1M
	d)	**Define entity set.** An entity set is a collection of similar types of entities that share the same attributes.	CO 1	L1	1M
	e)	**Differentiate relational algebra and relational calculus.** Relational algebra is a **procedural** query language that specifies *how* to obtain the result by providing a sequence of operations.	CO 2	L4	1M
	f)	**List the different datatypes in SQL.** Datatypes in SQL are broadly categorized into: 1. Numeric Data Types 2. Character/String Data Types 3. Date and Time Data Types 4. Binary Data Types 5. Boolean Data Types 6. Miscellaneous/Specialized Types	CO 2	L1	1M
	g)	**Why do we use JOIN in SQL?** We use JOIN in SQL to combine rows from two or more tables based on a related column between them.	CO 2	L1	1M
	h)	**Define B-Tree.** A B-tree is a self-balancing tree data structure designed to maintain sorted data and allow efficient searches, insertions, and deletions, particularly optimized for systems that store large amounts of data on disk.	CO 3	L1	1M
	i)	**What is meant by functional dependency.** A functional dependency (FD) is a constraint between two sets of attributes in a database table. It is represented as $X \rightarrow Y$, meaning X functionally determines Y.	CO 3	L1	1M
	j)	**Define Normal Form.** A Normal Form is a set of rules or conditions used in database design to organize data in tables. The primary goal of achieving normal forms is to reduce data redundancy, eliminate anomalies (like insertion, update, and deletion anomalies), and improve data integrity and consistency.	CO 3	L1	1M
	k)	**List various types of failures occur in transaction.** Various types of failures occur in transaction: 1. Transaction Failure 2. System Crash/Failure 3. Disk Failure (or Media Failure/Data Transfer Failure)	CO 4	L1	1M
	l)	**Define ROLLBACK.** ROLLBACK is a command in database management systems that undoes all changes made during the current transaction, effectively restoring the database to its state before the transaction began. It is used to ensure data integrity when errors occur or when a transaction needs to be aborted.	CO 4	L1	1M
	m)	**What is meant by timestamp?** In a database, a timestamp is a data type or a unique identifier assigned to a transaction or data record, typically representing the exact date and time when an event occurred	CO 4	L1	1M

n) **Define shadow paging.** CO 4 L1 1M

Shadow paging is a database recovery technique that uses two-page tables (current and shadow) to ensure atomicity and durability by applying changes to new pages and updating pointers only upon commit, or reverting to the shadow table on abort/crash.

Unit-I

2 a) **Explain the detailed system architecture of DBMS with a neat diagram.** CO 1 L2 7M

The Database System Environment:

a. Database Structure:

- A database system is partitioned into modules, each dealing with specific responsibilities of the overall system.
- The functional components of a database can be divided into:
  - Query processor components
  - Storage manager components
  - Disk Storage for the storage of data.
- These components are accessed by different types of users in the database environment.

b. Query Processor:

The Query Processor contains the following components:

- DDL Interpreter: Interprets DDL (Data Definition Language) statements and converts them into low-level data.
- DML Compiler: Translates DML (Data Manipulation Language) statements into low-level instructions that the query evaluation engine understands.
- Compiler and Linker: Converts information from user format to machine-understandable language. It also converts DML statements embedded in application programs to normal instructions by interacting with the DML compiler.
- Query Evaluation Engine: Executes low-level instructions generated by the DML compiler.

c. Storage Manager

- It is important because a database typically requires a large amount of storage space.
- It is a program module that provides an interface between low-level data stored in the database and the application programs and queries submitted to the system.
- It includes the following components:
  - Authorization and Integrity Manager: Tests for the satisfaction of integrity constraints and checks the authority of users to access data.
  - Transaction Manager: Ensures that the database remains in a consistent state when system failures occur and that concurrent transaction executions proceed without conflicting.
  - File Manager: Manages the allocation of storage space on disk and data structures used to store information.
  - Buffer Manager: Responsible for fetching data from disk into main memory.

d. Disk Storage

Disk storage includes the following components:

- Data Files: Stores the database itself.
- Data Dictionary: Stores metadata about the structure of the database. The data dictionary is used heavily, so good designing and implementation of the dictionary must be efficient.
- Indices: Provide fast access to data items that hold particular values.
- Statistical Data: Stores statistical information about the data in the database. This information is used by the Query processor to select efficient ways to execute a query.

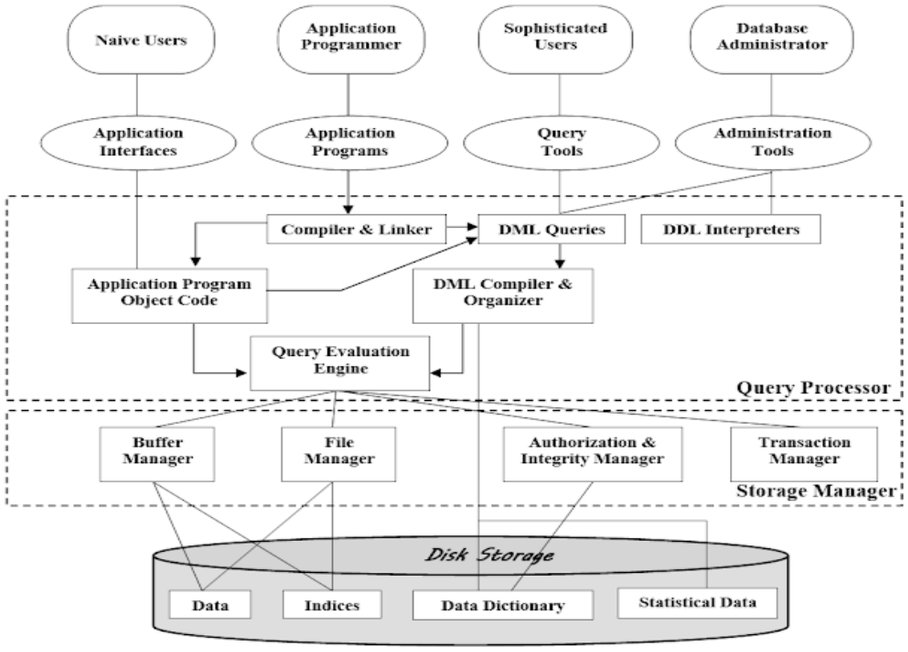
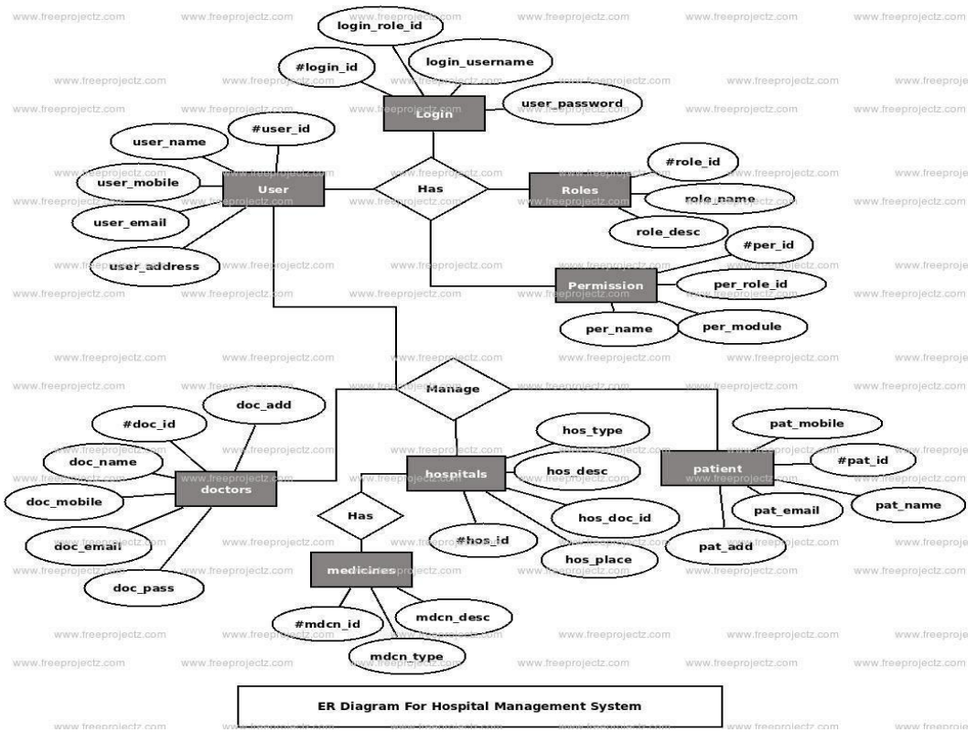


Figure: System Architecture

b) **Construct an ER diagram for a hospital with a set of patients and a set of medical doctors.** CO 1 L3 7M



(OR)

3 a) **Explain Three level schema architecture in detail.** CO 1 L2 7M

Three-Tier Client-Server Architecture

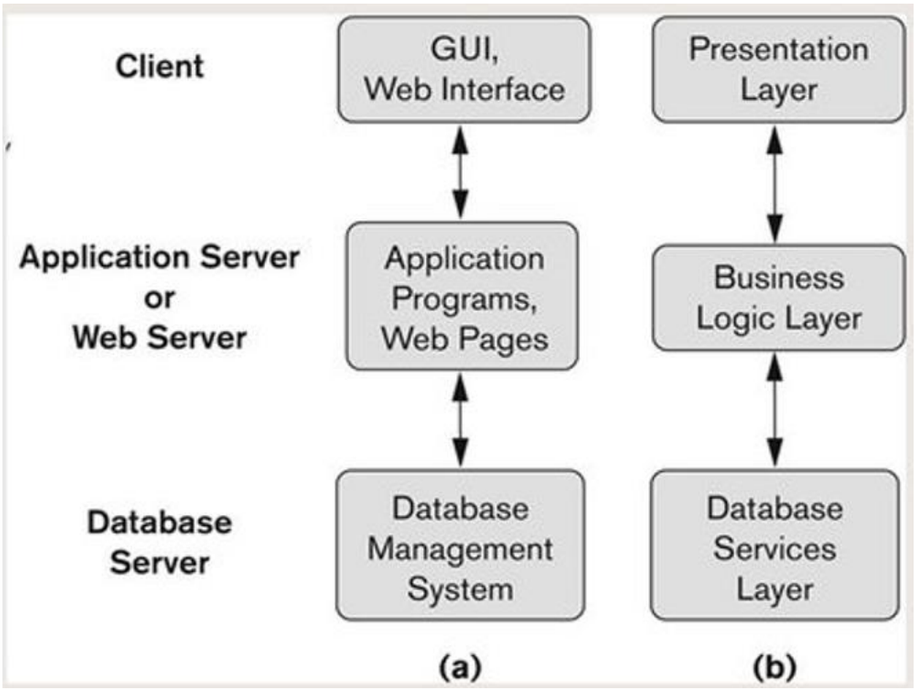
The three-tier client-server architecture is a software design pattern that logically and physically separates an application into three distinct layers, enhancing scalability, security, and maintainability.

1. **Presentation Tier (Client Layer):** This is the user interface layer, responsible for displaying information to the user and capturing their input. Examples include web browsers, desktop applications, or mobile apps. It typically contains no business logic or direct database access.
2. **Application Tier (Business Logic Layer / Middle Tier):** Acting as an intermediary, this tier hosts the core business logic, application programs, and web pages. It processes client requests, applies business rules, and communicates with the data tier. It can be an application server or web server and also handles user authorization and session management.
3. **Data Tier (Database Services Layer):** This lowest tier manages and stores all the application's data, typically comprising a Database Management System (DBMS). It provides data storage, retrieval,

integrity, and transaction management services to the application tier. Clients never directly access this tier.

Advantages:

- **Scalability:** Each tier can be scaled independently to handle increased load (e.g., adding more application servers).
- **Security:** The application tier acts as a buffer, preventing direct client access to the database and enabling centralized authorization.
- **Maintainability & Flexibility:** Changes in one tier (e.g., UI update) have minimal impact on others, simplifying development and upgrades.
- **Performance:** Workload distribution across tiers can optimize overall system performance.



b) **Discuss relationship types, sets, roles, and structural constraints in the context of ER modeling.** CO 1 L2 7M

Relationship Types, Sets, Roles, and Structural Constraints in ER Modeling

Relationship Type: A relationship type represents an association between two or more entities. It defines the type of logical connection that exists between instances of entity types. For example, 'WORKS_FOR' is a relationship type between 'EMPLOYEE' and 'DEPARTMENT' entities.

Relationship Set: A relationship set is a particular occurrence of a relationship type. It is the collection of all relationship instances that exist at a given point in time. For example, the 'Works_for' relationship set would contain individual links between specific employee entities (e.g., e1, e2) and department entities (e.g., d1, d2).

Roles: Role names signify the role that a participating entity plays in each relationship. Role names are essential, especially when the same entity type participates more than once in a relationship type in different roles. Such relationship types are called Recursive Relationships. An example is an 'EMPLOYEE' entity participating in a 'SUPERVISION' relationship type where one employee is a 'supervisor' and another is a 'supervisee'.

Structural Constraints on Relationship Types: Structural constraints define the rules that govern the participation of entities in a relationship type. These are primarily categorized into:

1. **Cardinality Ratios:**
    - This specifies the maximum number of relationship instances that an entity can participate in.
    - Possible cardinality ratios include:
        - **1:1 (One-to-One):** An entity from one set can be associated with at most one entity from the other set.

Example: an 'EMPLOYEE' can manage only one 'DEPARTMENT', and a 'DEPARTMENT' can have only one 'manager'.

- **1:N (One-to-Many):** An entity from one set can be associated with multiple entities from the other set, but an entity from the second set can be associated with at most one from the first.
- **N:1 (Many-to-One):** Multiple entities from one set can be associated with at most one entity from the other set, but an entity from the second set can be associated with multiple from the first.
- **M:N (Many-to-Many):** Multiple entities from one set can be associated with multiple entities from the other set. Example: 'Employee' 'Works_on' 'Project' where an employee can work on multiple projects, and a project can have multiple employees.

2. **Participation Constraints:**
   - This specifies whether the existence of an entity depends on its being related to another entity via the relationship type. There are two types:
     - **Total Participation:** If every entity in the total set of employee entities must be related to a department via a 'works-for' relationship, it is called total participation. This is also called Existence Dependency. For example, if every 'Employee' must work for a 'Department'.
     - **Partial Participation:** If some or part of the set of entities are related via the relationship, but not necessarily all. For instance, in a 'manages' relationship, not every 'Employee' is expected to manage a 'Department'.

Unit-II

4 a) **Explain about Tuple Relational Calculus (TRC).** CO 2 L3 8M

Tuple Relational Calculus (TRC) is a powerful, non-procedural query language used in relational database management systems (RDBMS). Unlike procedural languages (like Relational Algebra) that specify *how* to retrieve data, TRC focuses on *what* data to retrieve by defining the characteristics of the desired result set.

- **Non-Procedural Nature:** TRC is a declarative language, meaning users only specify the conditions that the desired tuples (rows) must satisfy, rather than providing a step-by-step process for data retrieval. The DBMS is then responsible for figuring out the most efficient way to obtain those results.
- **Tuple Variables:** TRC operates on tuple variables, which represent entire rows or records in a database table. A tuple variable iterates through each row of a specified relation (table) to check if a given condition (predicate) is true or false for that row.
- **Basic Syntax:** The general form of a TRC query is: { t | P(t) }
  - t: Represents the resulting tuple (row) variable.
  - P(t): This is a logical formula or predicate that describes the conditions that the tuples in the result must satisfy. The curly braces {} indicate that the expression is a set of tuples.
- **Logical Connectives:** The predicate P(t) can include various conditions combined using logical operators such as:
  - ∧ (AND): Both conditions must be true.
  - ∨ (OR): At least one condition must be true.
  - ¬ (NOT): Negation of a condition.
- **Quantifiers:** TRC utilizes quantifiers to express more complex conditions:
  - ∃ (Existential Quantifier - "there exists"):
    Specifies that at least one tuple satisfies the condition. For example, ∃ t ∈ r (Q(t)) means "there exists a tuple t in relation r such that predicate Q(t) is true."
  - ∀ (Universal Quantifier - "for all"):
    Specifies that a condition must be true for all tuples. For example, ∀ t ∈ r (Q(t)) means Q(t) is true "for all" tuples in relation r.
- **Examples:**
  - To retrieve the names of all employees earning more than $50,000 per year from an "Employees" table:
    { t.Name | Employees(t) ∧ t.Salary > 50000 }
  - To retrieve students enrolled in 'CS' course:
    { t | t ∈ STUDENT ∧ t.Course = 'CS' }

While powerful, TRC can sometimes be more challenging to write and understand compared to SQL. However, it serves as a fundamental theoretical basis for relational query languages and is useful in areas like formal verification of database schemas and academic research.

b) **Explain SQL aggregate operators with examples.** CO 2 L2 6M

SQL aggregate operators are functions that perform a calculation on a set of values (a column) and return a single summary value. These are frequently used with the GROUP BY clause to perform calculations on subsets of rows.
Here are the most common SQL aggregate operators with examples:

1. **COUNT()**:
   - **Purpose**: Returns the number of rows that match a specified criterion.
   - **Syntax**: COUNT(column_name) or COUNT(*) or COUNT(DISTINCT column_name)
   - **Examples**:
     - SELECT COUNT(*) FROM Employees; (Counts all rows in the Employees table)
     - SELECT COUNT(DISTINCT Department) FROM Employees; (Counts the number of unique departments)
     - SELECT COUNT(EmployeeID) FROM Employees WHERE Salary > 50000; (Counts employees with salary over 50,000)
2. **SUM()**:
   - **Purpose**: Returns the total sum of a numeric column.
   - **Syntax**: SUM(numeric_column_name)
   - **Example**:
     - SELECT SUM(Salary) FROM Employees; (Calculates the total sum of all employee salaries)
     - SELECT Department, SUM(Salary) FROM Employees GROUP BY Department; (Calculates the total salary for each department)
3. **AVG()**:
   - **Purpose**: Returns the average value of a numeric column.
   - **Syntax**: AVG(numeric_column_name)
   - **Example**:
     - SELECT AVG(Salary) FROM Employees; (Calculates the average salary of all employees)
     - SELECT Department, AVG(Salary) FROM Employees GROUP BY Department; (Calculates the average salary for each department)
4. **MIN()**:
   - **Purpose**: Returns the smallest value in a selected column.
   - **Syntax**: MIN(column_name)
   - **Example**:
     - SELECT MIN(Salary) FROM Employees; (Finds the lowest salary among all employees)
     - SELECT Department, MIN(Salary) FROM Employees GROUP BY Department; (Finds the lowest salary in each department)
5. **MAX()**:
   - **Purpose**: Returns the largest value in a selected column.
   - **Syntax**: MAX(column_name)
   - **Example**:
     - SELECT MAX(Salary) FROM Employees; (Finds the highest salary among all employees)
     - SELECT Department, MAX(Salary) FROM Employees GROUP BY Department; (Finds the highest salary in each department)

(OR)

5 a) **Explain binary relational operations with examples.** CO 2 L2 8M

Binary relational operations are fundamental operations in relational algebra that take two relations (tables) as input and produce a new relation as output. They are used to combine or compare data from multiple sources.
Here are the key binary relational operations with examples:

1. **Union (∪)**
   - **Purpose:** Combines all tuples (rows) from two union-compatible relations. Union-compatible means both relations must have the same number of columns, and corresponding columns must have compatible data types. Duplicate tuples are eliminated in the result.
   - **Example:** Let Employees table have columns (EmployeeID, Name, City) and Contractors table have (ContractorID, Name, City). SELECT EmployeeID, Name, City FROM Employees UNION SELECT ContractorID, Name, City FROM Contractors; This would return a list of all unique individuals (employees and contractors) with their IDs, names, and cities.
2. **Intersection (∩)**
   - **Purpose:** Returns only the tuples (rows) that are present in *both* union-compatible relations.
   - **Example:** Let CustomersInUSA table have (CustomerID, Name, City) and CustomersWithOrders table have (CustomerID, Name, City). SELECT CustomerID, Name, City FROM CustomersInUSA

INTERSECT SELECT CustomerID, Name, City FROM CustomersWithOrders; This would return customers who are both located in the USA and have placed orders.

3. **Set Difference (MINUS or EXCEPT)**
   - **Purpose:** Returns the tuples (rows) that are present in the first union-compatible relation but *not* in the second. (Often referred to as EXCEPT in SQL Server and MINUS in Oracle).
   - **Example:** Let AllProducts table have (ProductID, ProductName) and SoldOutProducts table have (ProductID, ProductName). SELECT ProductID, ProductName FROM AllProducts EXCEPT SELECT ProductID, ProductName FROM SoldOutProducts; This would return a list of products that are currently in stock (i.e., not sold out).
4. **Cartesian Product (× or CROSS JOIN)**
   - **Purpose:** Combines every tuple from the first relation with every tuple from the second relation. If relation R1 has 'n' tuples and R2 has 'm' tuples, the Cartesian Product will have 'n * m' tuples. This operation is typically used as a precursor to a JOIN operation where specific conditions are then applied.
   - **Example:** Let Departments table have (DeptID, DeptName) and Locations table have (LocID, City). SELECT * FROM Departments CROSS JOIN Locations; This would produce a table where every department is paired with every location, regardless of any actual relationship between them.
5. **Join (⋈)**
   - **Purpose:** Combines tuples from two relations based on a specified condition (join predicate). It is essentially a Cartesian Product followed by a selection operation. There are various types of joins (e.g., Inner Join, Left Join, Right Join, Full Outer Join), each with different rules for handling matching and non-matching rows.
   - **Example (Inner Join):** Let Employees table have (EmployeeID, Name, DeptID) and Departments table have (DeptID, DeptName). SELECT E.Name, D.DeptName FROM Employees E JOIN Departments D ON E.DeptID = D.DeptID; This would return employee names along with the names of their respective departments, only for employees who are assigned to a department that exists in the Departments table.

b) **Consider following relations and write SQL queries for given statements. Assume suitable constrains.** CO 2 L3 6M

Instructor(ID, Name, Dept_name , Salary)
Teaches(ID, Course_id, Sec_id, Semester(even/odd),Year)

i) Find the average salary of the instructors in computer department.
ii) Find the number of instructors in each department who teach a course in even semester of 2024.
iii) Find the names of instructor with salary amounts between 30000 and 50000.

Relations:
- Instructor(ID, Name, Dept_name, Salary)
- Teaches(ID, Course_id, Sec_id, Semester(even/odd), Year)

Constraints Assumption:
- ID in Instructor is the primary key and corresponds to ID in Teaches.
- Dept_name in Instructor is a string.
- Salary is a numeric value.
- Semester stores 'even' or 'odd'.
- Year stores a numeric year.

i) Find the average salary of the instructors in the computer department.

```
SELECT AVG(Salary) FROM Instructor WHERE Dept_name = 'Computer';
```

ii) Find the number of instructors in each department who teach a course in even semester of 2024.

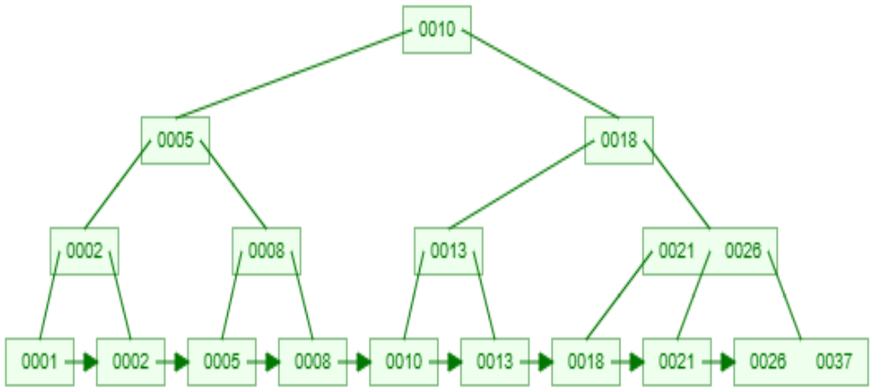
```
SELECT I.Dept_name, COUNT(DISTINCT I.ID) AS NumberOfInstructors
FROM Instructor AS I
JOIN Teaches AS T ON I.ID = T.ID
WHERE T.Semester = 'even' AND T.Year = 2024 GROUP BY I.Dept_name;
```

iii) Find the names of instructor with salary amounts between 30000 and 50000.

```
SELECT Name FROM Instructor WHERE Salary BETWEEN 30000 AND
50000;
```

Unit-III

6 a) Construct B+ tree of order 3 with the following values 1,2,5,8,10,13,18,21,26,37. CO 3 L3 7M



b) Explain about BCNF and list the differences between BCNF and 3NF. CO 3 L2 7M

Boyce-Codd Normal Form (BCNF)
BCNF is a normal form for relational databases. A relation is in BCNF if for every functional dependency (FD)
X→Y that holds in the relation, X is a superkey of that relation. This means that in a BCNF relation, all determinants are candidate keys.
Example: Consider a table with attributes "Sid", "emailid", "cid", and "marks".

- Candidate keys are

(Sid, cid) and (cid, emailid).

- Functional dependencies include

Sid → emailid and emailid → Sid.

- (Sid, cid) → marks.
- (cid, emailid) → marks.

This table is in 3NF but not in BCNF. This is because
Sid and emailid are determinants, but they are not superkeys on their own. To make this table BCNF, it needs to be split. For instance, it can be decomposed into two tables:

1. (sid, emailid): Here, sid decides emailid, and emailid decides sid, and both are candidate keys.
2. (sid, cid, Marks): Here, (sid, cid) is a candidate key.

After this decomposition, both tables are in BCNF because all their determinants are candidate keys.

Feature	Third Normal Form (3NF)	Boyce-Codd Normal Form (BCNF)
Definition	A relation R is in 3NF if for every FD XrightarrowA in R, either: (a) X is a superkey of R, OR (b) A is a prime attribute of R.	A relation R is in BCNF if for every FD XrightarrowA in R, X **must** be a superkey of R.
Strictness	Less strict than BCNF.	Stricter than 3NF.
Condition (b)	Allows a non-superkey to functionally determine a prime attribute.	Disallows a non-superkey to functionally determine a prime attribute.
Anomaly Handling	Addresses most update anomalies, but some may still exist if condition (b) is met.	Addresses all update anomalies related to functional dependencies.
Dependency Preservation	Always guarantees dependency preservation during decomposition.	May not always preserve all functional dependencies during decomposition.
Lossless Join	Always guarantees lossless join decomposition.	Always guarantees lossless join decomposition.
Example	A relation can be in 3NF but not in BCNF if a non-superkey determines a prime attribute (e.g., TEACH relation where instructor (non-superkey) determines course (prime attribute)).	A relation in BCNF is always in 3NF.

(OR)

7 a) Explain the concepts of Functional dependency with suitable example and its operations like closure. CO 3 L2 7M

Functional Dependencies (FDs)

A **Functional Dependency (FD)** is a constraint that describes a relationship between attributes in a relational database. It states that the value of one set of attributes (the **determinant**, X) uniquely determines the value of another set of attributes (the **dependent**, Y). This is denoted as X→Y.

Definition: An FD X→Y holds in a relation R if, for any two tuples t1 and t2 in any valid instance of R, whenever t1[X]=t2[X], then it must also be true that t1[Y]=t2[Y].

Example: Consider a STUDENT relation with attributes {StudentID, StudentName, Major}. A functional dependency StudentID → StudentName means that if two students have the same StudentID, they must also have the same StudentName. This reflects a real-world constraint where a student's ID uniquely identifies their name.

Operations: Closure of a Set of Attributes (X+)

The **closure of a set of attributes** X **with respect to a set of FDs** F, denoted as X+, is the set of all attributes that are functionally determined by X. In essence, it tells you all the attributes whose values you can determine if you know the values of the attributes in X.

Purpose: The closure of a set of attributes is crucial for:

1. **Finding Candidate Keys:** If the closure of a set of attributes X contains all attributes of the relation, then X is a **superkey**. If X is also minimal (no proper subset of X is a superkey), then X is a **candidate key**.
2. **Normalization:** It helps in identifying and resolving anomalies during the database normalization process.

Algorithm for Computing X+:

To compute the closure of a set of attributes X under a given set of FDs F:

1. **Initialization:** Start with X+=X.
2. **Iteration:** Repeatedly apply the following rule until no new attributes can be added to X+: For every functional dependency A→B in F: If all attributes in A are already present in X+, then add all attributes in B to X+.
3. **Result:** The final set X+ is the closure of X.

Example of Computing X+:

Let's use a relation R (A, B, C, D, E) and a set of FDs F = {A→BC, C→D, B→E}. We want to find the closure of A, i.e., A+.

1. **Initialize:** A+={A}**Error! Filename not specified.**
2. **Iterate:**
   - From F, consider A→BC. Since A is in A+, add B and C to A+. A+ becomes {A,B,C}.
   - From F, consider C→D. Since C is now in A+, add D to A+. A+ becomes {A,B,C,D}.
   - From F, consider B→E. Since B is now in A+, add E to A+. A+ becomes {A,B,C,D,E}.
3. **Check for Stability:** No new attributes can be added in subsequent iterations.
4. **Final Result:** A+={A,B,C,D,E}. Since A+ contains all attributes of the relation R, A is a superkey for R.

b) Consider schema R = (A, B, C, G, H, I) and the set F of functional dependencies {A→ B, A→C, CG→H, CG→I, B→H}. Compute the candidate keys of the schema. Compute the closure of the same. CO 3 L4 7M

Given Schema: R= (A, B, C, G, H, I) **Given Functional Dependencies (FDs):**
F={A→B, A→C, CG→H, CG→I, B→H}

Computing Closure of Attributes (X+)

We'll use the algorithm for computing X+ as described in the Canvas to find superkeys, which are the basis for candidate keys. A superkey is a set of attributes whose closure contains all attributes in the schema R.

1. Closure of A (A+):

- **Initialize:** A+={A}
- **Iterate:**
  - Using A→B: A+= {A, B}
  - Using A→C: A+= {A, B, C}
  - Using B→H: A+= {A, B, C, H}
- **Result:** A+= {A, B, C, H} *This is not a superkey as it does not contain all attributes of R (missing G, I).*

2. Closure of G (G+):

- **Initialize:** G+={G}
- **Result:** G+={G} *This is not a superkey.*

3. Closure of AG ((AG)+):

- **Initialize:** (AG) += {A, G}
- **Iterate:**
  - Using A→B: (AG)+= {A, G, B}
  - Using A→C: (AG)+= {A, G, B, C}
  - Using B→H: (AG)+= {A, G, B, C, H}
  - Using CG→H: (C and G are in (AG)+, H is already there)
  - Using CG→I: (C and G are in (AG)+, add I) (AG)+= {A, G, B, C, H, I}

- **Result:** (AG)+= {A, B, C, G, H, I} *Since (AG)+ contains all attributes of R, AG is a **superkey**.*

Computing Candidate Keys

A **candidate key** is a minimal superkey. This means that no proper subset of a candidate key can also be a superkey. We need to check if the superkey AG is minimal.

- We already checked A+ and G+ above. Neither A nor G alone is a superkey.

Since AG is a superkey, and none of its proper subsets (A or G) are superkeys, AG is a minimal superkey.

Therefore, the only candidate key for the schema R is {A, G}.

Unit-IV

8 a) Define locking protocol. Explain the Two-Phase Locking protocol with an example. CO 4 L2 7M

A locking protocol is a set of rules used in database concurrency control to manage simultaneous access to shared data items by multiple transactions. Its primary goal is to ensure the serializability of schedules, meaning that concurrent transactions produce the same result as if they were executed serially, thereby maintaining database consistency.

The **Two-Phase Locking (2PL) Protocol** is a widely used concurrency control protocol that guarantees serializability. It operates in two distinct phases for each transaction:

1. **Expanding (Growing) Phase:**
   - During this phase, a transaction can acquire new locks on data items, but it cannot release any locks it currently holds.
   - If a transaction needs to upgrade a lock (e.g., from a shared lock to an exclusive lock), this must also occur during the expanding phase.
2. **Shrinking Phase:**
   - Once a transaction releases its first lock, it enters the shrinking phase.
   - In this phase, the transaction can release existing locks, but it cannot acquire any new locks.
   - Downgrading a lock (e.g., from an exclusive lock to a shared lock) must be done during this phase.

Example (Conceptual, based on Figure 21.3 from UNIT-4 Ch-2.pptx):

Consider two transactions, T1 and T2, accessing data items X and Y.

- **Transaction T1:**
  - read_lock(Y) (Expanding Phase: Acquires lock)
  - read_item(Y)
  - unlock(Y) (Shrinking Phase: Releases lock)
  - write_lock(X) (Violates 2PL, as it's trying to acquire a new lock in the shrinking phase)
- **Transaction T2:**
  - read_lock(X) (Expanding Phase)
  - read_item(X)
  - write_lock(Y) (Expanding Phase)
  - read_item(Y)
  - unlock(X) (Shrinking Phase)
  - write_item(Y)
  - unlock(Y)

If T1 were to strictly follow 2PL, it would need to acquire all necessary locks (e.g., on Y and X) before releasing any, ensuring that its write_lock(X) operation occurs before unlock(Y). By enforcing these two phases, 2PL ensures that transactions do not interfere with each other in ways that would lead to non-serializable schedules.

b) Differentiate serializable schedule, recoverable schedule and strict schedule. CO 4 L4 7M

Differentiating Schedule Types in Concurrency Control

In database concurrency control, schedules define the order of operations from interleaved transactions. Their "goodness" is characterized by properties that ensure database consistency and correct behavior.

1. **Serializable Schedule:**
   - **Definition:** A schedule is **serializable** if its effect on the database is equivalent to some serial (non-interleaved) execution of the same set of transactions.
   - **Implication:** It guarantees that the final state of the database will be consistent, as if transactions ran one after another. This is the ultimate goal for correct concurrent execution.
2. **Recoverable Schedule:**
   - **Definition:** A schedule is **recoverable** if, for every pair of transactions Ti and Tj where Tj reads a data item previously written by Ti, Ti must commit before Tj commits.
   - **Implication:** This property prevents "cascading rollbacks." If Ti aborts after Tj has read its uncommitted data, Tj would also have to abort, and potentially other transactions that read from Tj. Recoverability ensures that a transaction doesn't commit based on uncommitted data from

another transaction that might later fail.

3. **Strict Schedule:**
    - **Definition:** A schedule is **strict** if a transaction is not allowed to read or write a data item X until the transaction that last wrote X has committed.
    - **Implication:** This is a stricter condition than recoverability. It prevents "dirty reads" (reading uncommitted data) and "dirty writes" (overwriting uncommitted data). Strict schedules are highly desirable because they simplify recovery significantly, as no cascading rollbacks are possible, and all visible data is guaranteed to be committed.

Key Differences/Relationship:

These properties form a hierarchy: **Strict Schedules ⟹ Recoverable Schedules ⟹ Serializable Schedules**.

- All strict schedules are recoverable.
- All recoverable schedules are serializable.
- However, a serializable schedule is not necessarily recoverable, and a recoverable schedule is not necessarily strict. The stricter properties (recoverable, strict) impose additional constraints primarily for ease and correctness of recovery from transaction failures.

(OR)

9 a) Explain about ACID properties with an example. CO 4 L2 7M

The **ACID properties** are a set of fundamental principles that guarantee the reliability of database transactions. They ensure that data remains valid and consistent even in the event of errors, power failures, or concurrent access.

Here's an explanation of each property with an example, drawing from the uploaded materials:

1. **Atomicity:**
    - **Definition:** A transaction is treated as a single, indivisible unit of work. It is either fully completed (all its operations succeed) or entirely aborted (none of its operations take effect). There is no "partial" completion.
    - **Example:** In a money transfer from Account A to Account B, either money is debited from A *and* credited to B, or if any part fails, neither operation occurs. The transaction is atomic.
2. **Consistency:**
    - **Definition:** A transaction must bring the database from one valid and consistent state to another. It ensures that all predefined rules, constraints, and integrity properties of the database are maintained.
    - **Example:** Before the transfer, the sum of balances in A and B is X. After a successful transfer, the sum of balances in A and B must still be X. The transaction maintains the consistency rule that money is neither created nor destroyed.
3. **Isolation:**
    - **Definition:** Concurrent execution of multiple transactions should not interfere with each other. Each transaction appears to execute in isolation, as if it were the only transaction running on the system. Updates by one transaction are not visible to other transactions until the first transaction commits.
    - **Example:** If Account A has $100, and Transaction T1 debits $50 while Transaction T2 simultaneously tries to read Account A's balance, T2 will either read $100 (before T1 commits) or $50 (after T1 commits), but never an intermediate, uncommitted value.
4. **Durability (or Permanency):**
    - **Definition:** Once a transaction has been committed, its changes are permanent and will survive any subsequent system failures (e.g., power outages, crashes). The committed data is written to stable storage.
    - **Example:** After the money transfer from A to B is committed, even if the database system crashes immediately afterward, the new balances in Account A and Account B will persist and be available when the system recovers.

b) Discuss recovery techniques based on immediate update with example. CO 4 L2 7M

Here's a discussion on recovery techniques based on immediate update, suitable for a 7-mark answer:

Recovery Techniques Based on Immediate Update

Immediate update is a database recovery approach where a transaction's updates are written directly to the database disk *before* the transaction reaches its commit point. This contrasts with deferred update, where changes are buffered and written only upon commit. Because updates are made immediately, both **undo** and **redo** operations may be necessary for recovery after a system failure.

Core Concept: Since changes are written to disk immediately, a failure can occur at any point.

- If a transaction has updated the database but not yet committed, its changes must be **undone** (rolled back) to restore the database to its state before the transaction began.
- If a transaction has committed, but some of its final updates (which were written to memory buffers but not yet to disk) were lost due to the crash, these changes must be **redone** to ensure durability.

Key Algorithms and Mechanisms:

1. **Write-Ahead Logging (WAL):** This is fundamental to immediate update. All database modifications (old and new values) must be recorded in the system log on stable storage *before* the actual database items are updated on disk. This ensures that sufficient information is available to either undo or redo operations after a crash.
2. **Undo/Redo Algorithm:** This is the most common algorithm for immediate update.
    - **Recovery Process:** When the system recovers from a crash, the recovery manager examines the log:
        - **Undo:** For any transaction that has a [start_transaction, T] log entry but no [commit, T] entry (i.e., active or aborted transactions), its changes are undone by tracing backward through the log and restoring old_values.
        - **Redo:** For any transaction that has a [commit, T] entry in the log, its changes are redone by tracing forward through the log and applying new_values to ensure all committed changes are permanently reflected.
    - **Example:**
        - Suppose Transaction T1 updates item X from 10 to 20, and then the system crashes before T1 commits. The log would contain [write_item, T1, X, 10, 20]. Upon recovery, the system sees no commit record for T1, so it performs an **undo** operation, setting X back to 10.
        - Suppose Transaction T2 updates item Y from 30 to 40 and commits, but the updated block for Y was still in a main memory buffer and not yet written to disk. The log contains [write_item, T2, Y, 30, 40] and [commit, T2]. Upon recovery, the system sees the commit record for T2, so it performs a **redo** operation, ensuring Y is set to 40.
3. **Checkpointing:** To minimize the amount of work (scanning the log and performing undo/redo operations) required during recovery, checkpoints are periodically taken. A checkpoint records the state of the database at a certain point in time, reducing the portion of the log that needs to be processed during recovery.

Immediate update with the Undo/Redo algorithm provides robust recovery capabilities by ensuring that committed transactions are durable and uncommitted transactions are fully undone, maintaining database consistency.

SIGNATURE OF THE FACULTY **SIGNATURE OF THE HEAD OF THE DEPARTMENT**