GTU DBMS Winter 2024 Solution

PREPZONE

GTU - DBMS Winter 2024 PAPER SOLUTION

Subject Code: 3130703 | **Subject Name:** DBMS | **Total Marks:** 70

Q.1 (a) Define following terms. (03 Marks)

i) Database Management System (DBMS)

- A **DBMS** is a collection of interrelated data and a set of programs to access that data.
- It's a **software system** that enables users to define, create, maintain, and control access to the database.
- The primary goal is to provide a way to store and retrieve database information conveniently and efficiently.
- It acts as an **intermediary** between the users and the database.
- **Examples** include MySQL, Oracle, PostgreSQL, and Microsoft SQL Server.

GTU PREPZONE

ii) Instance

- An **instance** is the data contained in the database at a particular **moment in time**.
- It represents the actual **content** of the database at that specific instant.
- The structure (schema) of the database is static, but the instance changes whenever data is inserted, deleted, or updated.
- It is also known as the **state** of the database.
- For example, if a table has 10 rows right now, the collection of these 10 rows is the instance.

iii) Logical Data Independence

- **Logical Data Independence** refers to the ability to change the **conceptual schema** (logical structure) without affecting the **external schemas** (user views).
- It means users or application programs are **shielded** from changes made to the database's logical structure.
- This is achieved by mapping the external view to the conceptual schema.
- It is considered **more difficult** to achieve than physical data independence.
- For example, adding a new column to a table (conceptual schema change) should not affect applications that do not use that column.

Q.1 (b) List and explain categories of database users. Explain roles and responsibility of DBA (04 Marks)

Database users can be broadly classified into several categories based on their interaction with the system.

1. **Application Programmers:**
   - They are computer professionals who write **application programs** to interact with the database.
   - These programs can be written in high-level languages and contain calls to the DBMS (e.g., SQL queries).
2. **Sophisticated Users:**
   - These users interact with the system without writing programs.
   - They typically use a **query language** (like SQL) to submit their requests directly.
   - Analysts and engineers fall into this category.
3. **Specialized Users:**
   - These are sophisticated users who write specialized database applications.
   - Examples include developing **CAD systems, knowledge-based systems**, or complex data analysis programs.
4. **Naive/End Users:**
   - They are unsophisticated users who access the database through **pre-written application programs**.
   - They use an interface like forms or menus and are unaware of the underlying DBMS details.
5. **Database Administrator (DBA):**
   - The DBA is a highly trained person responsible for the **overall control** of the database system.
   - Their role involves schema definition, storage structure, and access control.
6. **System Analysts:**
   - They determine the requirements of end-users and develop specifications for transactions.
7. **Casual Users:**
   - Users who access the database only **occasionally**, but they may require different information each time.

The **Database Administrator (DBA)** is the person or group responsible for managing and maintaining the database system.

GTU PREPZONE

1. **Schema Definition:** The DBA is responsible for creating the original database **schema** (structure) using the Data Definition Language (DDL).
2. **Storage Structure and Access Method Definition:** Deciding on the appropriate file organization, indexes, and physical storage parameters for the database.
3. **Schema and Physical Modification:** Making necessary changes to the database structure (schema and physical level) as organizational requirements evolve.
4. **Granting Authorization for Data Access:** The DBA controls which users can access what parts of the database and what operations they can perform (e.g., using **GRANT/REVOKE**).
5. **Routine Maintenance:** This includes ensuring the database is running efficiently, performing backups and recovery, and monitoring system performance.
6. **Performance Monitoring and Tuning:** Analyzing the database workload and optimizing the system to ensure good response time for all users.
7. **Developing Backup and Recovery Strategies:** Defining procedures to be followed in case of hardware or software failures to restore the database to a consistent state.
8. **Security Management:** Implementing and maintaining security policies to protect the database from unauthorized access and potential threats.

GTU PREPZONE

Q.1 (c) Compare the advantages of using a Database Management System (DBMS) over a file processing system. (07 Marks)

1. **Better Data Security:** DBMS offers centralized control with advance security features like encryption, authentication and authorization. It restrict unauthorized access and protect data.

2. **Reduced data redundancy:** stored data in structured and centralized way, minimizing duplication. It supports shared access across applications, eliminating repeated storage.

3. **Improved data consistency:** maintained by enforcing validation rules and constraints. It eans data is consistent(same) across all pplatforms.

4. **Improved data access:** provides efficient data access and retrieval mechanism that enables quick and easy data access.

5. **Improved data sharing:** Allows secure and simultaneous data sharing across systems and users.

6. **Improved data Integration:** combines data form various sources into unified view. Maintains integrity through referential, domain and entity constraints.

7. **Data Backup and recovery:** offers automated reliable backup and point-in-time recovery. Data can be restored after failures maintaining consistency and availability.

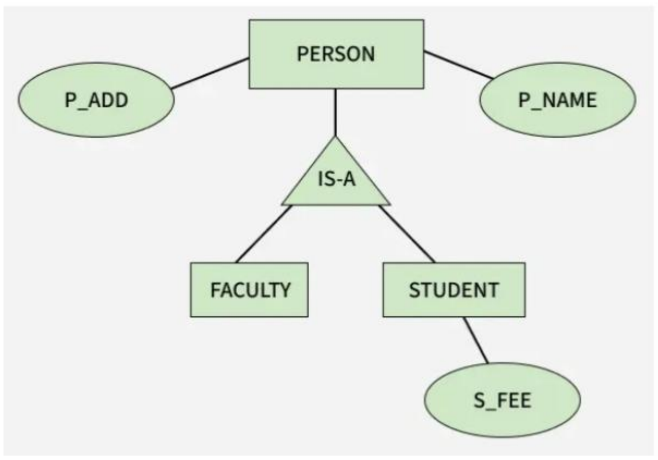
Q.2 (a) Explain generalization and specialization with neat diagram. (03 Marks)

Generalization and **Specialization** are concepts used in the Enhanced Entity-Relationship (E-R) model to structure entities based on their common and distinct properties.

Generalization (Bottom-Up Approach)

- **Definition:** It is a **bottom-up** process where two or more distinct lower-level **entity types** are combined to form a higher-level **generalized entity type** based on common features.
- **Concept:** It identifies common attributes and relationships among entities and groups them into a superclass.
- **Example:** ENTITY Car and ENTITY Truck can be generalized to a higher-level entity Vehicle because they share attributes like Registration_ID and Color.
- **Diagram:** The arrow points **from** the specialized entities **to** the generalized entity.
- **Mnemonic:** Think of it as summarizing or moving *up* the hierarchy.

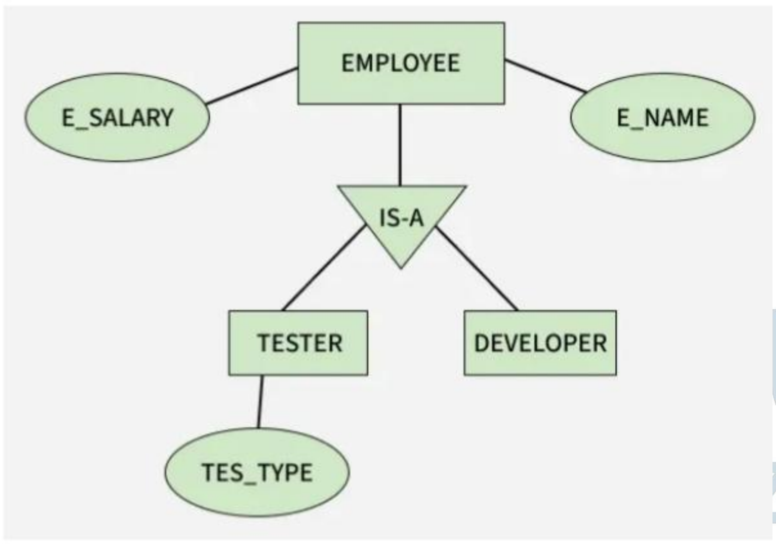
GTU PREPZONE



Generalization

Specialization (Top-Down Approach)

- **Definition:** It is a **top-down** process of defining a set of lower-level **specialized entity types** from a higher-level **generalized entity type**.
- **Concept:** It defines subclasses based on attributes specific to a subset of the higher-level entity's instances.
- **Example:** The ENTITY Employee can be specialized into Engineer, Accountant, and Manager based on their roles and unique attributes.
- **Diagram:** The arrow points **from** the generalized entity **to** the** specialized entities**.
- **Mnemonic:** Think of it as detailing or moving *down* the hierarchy.



Specialization

U
ZONE

Q.2 (b) Explain the following attributes: (04 Marks)

i. **Single value:** This type of attribute can hold only 1 value for specific entity instance.

Example: A persons date of birth will only be single value

ii. **Derived value:** The value of this attribute is not physically stored in database but is instead calculated or derived from other stored attributes.

Example: Persons age can be calculated from their stored date of birth.

iii. **Multi-valued:**This attribute can have multiple values associated with single entity instance.

Example: A person can have multiple phone numbers, several hobbies, or knows multiple languages.

iv. **Composite :** This attribute is combined from several smaller , simple attributes each with independent meaning.

Example: Address attribute might be broken down into street, city, state, zip code etc..

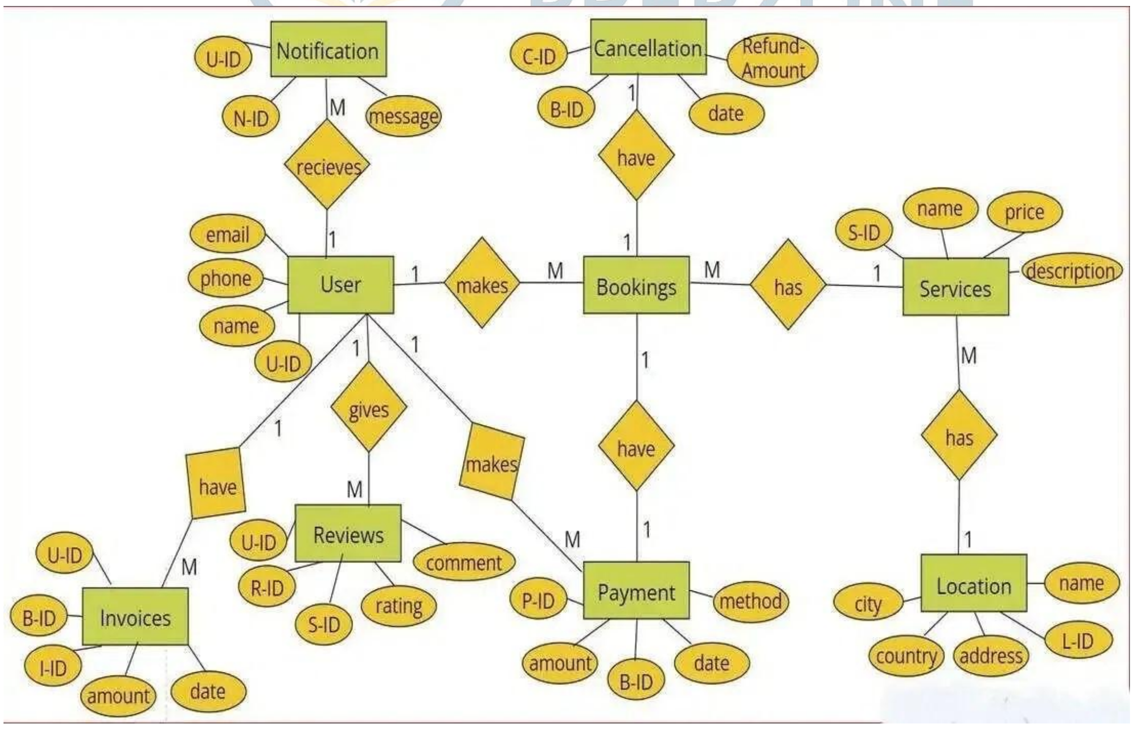
Q.2 (c) Draw an E-R diagram of following scenario. Make necessary assumptions and clearly note down the assumptions. (07 Marks)

The scenario is a **Municipal Corporation/any Bus reservation system**.

Assumptions:

1. **Users**: user_id, user_name, email, phone
2. **Bookings:** bookingid, userod, date, status
3. **Locations:** location-id, name, addresses, state, city, country
4. **Reviews:** review id, user id, rating comments
5. **Payment:** payment id, booking id, amount, payment date, payment method
6. **Cancellation/refund:** Cancellation/refund id, booking id, cancellation date, refund amount

GTU PREPZONE



OR Q.2 (c) Explain the concepts of strong entity and weak entity using real world example. (07 Marks)

1. Strong Entity Set

- **Definition:** A strong entity set is an entity that has a **primary key** and can be uniquely identified on its own.
- **Dependence:** It is **independent** of any other entity set in the database.
- **Representation (E-R Diagram):** It is represented by a **single rectangle**.
- **Primary Key:** Its primary key is formed using its **own attributes**.
- **Relationship:** It participates in identifying relationships as the **identifying entity**.
- **Example: EMPLOYEE** is a strong entity because it can be uniquely identified by the attribute Employee_ID (its primary key), regardless of any other entity.

2. Weak Entity Set

- **Definition:** A weak entity set is an entity that **does not have a primary key** of its own and depends on another entity for its unique identification.
- **Dependence:** It is **existence-dependent** on a strong entity set, called its *identifying entity*.
- **Representation (E-R Diagram):** It is represented by a **double rectangle**.
- **Partial Key:** It has a **partial key** (or discriminator), which is the set of attributes that can uniquely identify the entity within the *strong* entity it depends on.
- **Identifying Relationship:** The relationship connecting the weak entity set to the strong entity set is called the **identifying relationship**, which is represented by a **double diamond**.
- **Primary Key Formation:** Its primary key is formed by combining its **partial key** with the **primary key of its identifying strong entity**.
- **Real World Example:** The entity **DEPENDENT** is often a weak entity. A dependent (e.g., a child) cannot be uniquely identified without knowing which **EMPLOYEE** they are dependent on. The partial key might be *Dependent_Name*, and the full primary key would be (Employee_ID,Dependent_Name).

GTU PREPZONE

Q.3 (a) Explain the terms. (03 Marks)

i) Super Key

- A **Super Key** is a set of one or more attributes that, taken together, can **uniquely identify** a tuple (row) in a relation.
- It's any combination of attributes that includes a **candidate key**.
- The primary key is always a super key, but a super key is not necessarily a primary key.
- If a set of attributes K is a super key, any superset of K is also a super key.
- **Example:** In a table with attributes (ID,Name,Age), both (ID) and (ID,Name) are super keys, assuming ID is unique.

ii) Foreign Key

- A **Foreign Key** is a set of attributes in a relation (the referencing relation) that refers to the **primary key** or a **unique key** of another relation (the referenced relation).
- It is used to establish a **link** between two tables.
- The attribute values in the foreign key must appear in the primary key of the referenced table or be NULL.
- It is crucial for enforcing **referential integrity** in the database.
- **Example:** The attribute Dept_ID in an Employee table that references the ID in the Department table is a foreign key.

GTU PREPZONE

iii) Unique Key

- A **Unique Key** is an attribute or a set of attributes that **uniquely identifies** each tuple in a relation, similar to a primary key.
- **Difference from Primary Key:** A unique key can have a **NULL** value, whereas a primary key cannot (it must be NOT NULL).
- A relation can have **multiple** unique keys.
- It acts as a **candidate key** that is not chosen as the primary key.
- It ensures that all values in the column or columns are distinct (unique).

Q.3 (b) Explain trivial functional dependencies with suitable example. (04 Marks)

Trivial Functional Dependencies (FDs)

1. **Definition:** A functional dependency $X \rightarrow Y$ is considered **trivial** if the set of attributes Y is a **subset or equal** to the set of attributes X. ($Y \subseteq X$).

2. **Concept:** This type of dependency does not convey any extra information because the dependent attribute(s) are already part of the determinant attribute(s).
3. **Inherently True:** Trivial FDs are always true for **any relation instance** because knowing the value of X always means you inherently know the value of any part of X.
4. **Informational Value:** They are generally of **little interest** in normalization and database design because they state the obvious.
5. **Example 1:** Given a relation R(A,B,C). The FD (A,B)→A is trivial because A is a subset of (A,B).
6. **Example 2:** The FD (A,B)→(A,B) is also trivial (since Y=X).
7. **Non-Trivial:** A dependency X→Y is **non-trivial** if Y is *not* a subset of X (Y ⊆X).

Q.3 (c) Consider a relation R(A, B, C, D, E) with the following three functional dependencies. (07 Marks)

FD={A→BC;BC→D;C→E}

Find out the number of superkeys in the relation R.

GTU PREPZONE

1. Find the Candidate Key(s)

- **Attributes not on the Right Hand Side (RHS):** Check which attributes do not appear on the RHS of any FD.
  - RHS attributes: B,C,D,E.
  - Attributes not on RHS: **A**.
- Any candidate key must contain the attribute **A**. Let's check A+ (the closure of A).
- **A+ Calculation:**
  - Start with A+={A}.
  - Using A→BC: A+={A,B,C}.
  - Using BC→D: Since B,C∈A+, we add D. A+={A,B,C,D}.
  - Using C→E: Since C∈A+, we add E. A+={A,B,C,D,E}.
- Since A+ contains all attributes of R, A is a **Candidate Key (CK)**.
- **Check for other CKs:** Since A is the only attribute not on the RHS, and A itself is a CK, there can be no other CK.
- **The only Candidate Key is {A}.**

2. Find the Superkeys

- A **Super Key** is any set of attributes that contains a Candidate Key.

- Since the only Candidate Key is A, a Super Key must be any subset of R's attributes that contains A.
- The relation R has 5 attributes: {A,B,C,D,E}.
- The number of possible subsets of R is 25=32.
- We need to count the number of subsets that contain A. This is equivalent to finding the number of subsets of the remaining attributes {B,C,D,E} and adding A to each of them.
- The set of remaining attributes has 4 elements: S={B,C,D,E}.
- The number of subsets of S is 24=16.
- Each of these 16 subsets, when combined with A, forms a superkey.
- **Number of Superkeys** = Number of subsets of R−{A}=2(5−1)=24=* * 16* * .
- **The total number of superkeys is 16.**

OR Q.3 (a) Explain insertion and deleting anomalies with respect to normalization. (03 Marks)

i) Insertion Anomaly

GTU PREPZONE

- **Definition:** An insertion anomaly occurs when we cannot insert a new tuple (row) into a relation unless we also have a value for a related, but independent, piece of information.
- **Cause:** It happens when a relation schema is not fully normalized and contains a non-key attribute dependent on only a *part* of a composite primary key (**Violation of 2NF**).
- **Problem:** To add a new instance of one entity, we are forced to invent data for another entity.
- **Example:** In a non-normalized Employee_Dept table (EmpID,DeptName,DeptManager), we cannot add a new Department until an Employee is assigned to it, as EmpID is part of the primary key.
- **Solution:** Decompose the relation into smaller, more normalized relations (e.g., Employee and Department).

ii) Deletion Anomaly

- **Definition:** A deletion anomaly occurs when the deletion of a single tuple results in the **unintended loss of other critical information** stored in the same tuple.
- **Cause:** Like insertion anomaly, it arises from storing two different, non-fully dependent facts in the same relation (redundancy).
- **Problem:** Deleting the last instance of one entity may inadvertently remove all information about a related entity.

- **Example:** In the Employee_Dept table, if the last Employee in the 'Sales' department is deleted, the entire information about the 'Sales' department and its DeptManager is lost.
- **Solution:** Normalization (decomposition) separates these concepts into different tables, ensuring information is not lost during tuple deletion.

OR Q.3 (b) Explain Armstrong's axioms in detail. (04 Marks)

Armstrong's Axioms are a set of inference rules used to logically derive all functional dependencies (FDs) implied by a given set of FDs (known as the closure of the FD set). They are sound (only generate correct FDs) and complete (generate all correct FDs).

1. **Axiom of Reflexivity (Trivial FDs):**
   - **Rule:** If Y is a subset of X (Y⊆X), then X→Y holds.
   - **Explanation:** Knowing the attributes in set X automatically means you know the values of any subset Y of X. This covers the trivial functional dependencies.
2. **Axiom of Augmentation:**
   - **Rule:** If X→Y holds, then XZ→YZ holds for any set of attributes Z.
   - **Explanation:** Adding an extra attribute set Z to the determinant (X) and the dependent set (Y) does not invalidate the original dependency. If X determines Y, then X plus any extra attributes Z will still determine Y plus Z.
3. **Axiom of Transitivity:**
   - **Rule:** If X→Y holds and Y→Z holds, then X→Z also holds.
   - **Explanation:** This allows for chaining FDs. If X determines Y, and Y determines Z, then X indirectly determines Z.

Additional Derived Rules (Not primary axioms but used for convenience):

- **Decomposition (Additive):** If X→YZ, then X→Y and X→Z.
- **Union (Combining):** If X→Y and X→Z, then X→YZ.
- **Pseudotransitivity:** If X→Y and WY→Z, then WX→Z.

GTU PREPZONE

OR Q.3 (c) Given a relation R(P, Q, R, S, T) and Functional Dependency set FD={PQ→R,S→T}, determine whether the given R is in 2NF? If not convert it into 2 NF. (07 Marks)

1. Find the Candidate Key(s)

- **RHS Attributes:** R,T.
- **Attributes not on RHS:** P,Q,S. These must be part of any Candidate Key. Let's check the closure of PQS.
- (PQS)+:
  - Start with {P,Q,S}.
  - Using PQ→R: Add R. →{P,Q,S,R}.
  - Using S→T: Add T. →{P,Q,S,R,T}.
- Since (PQS)+=R, **PQS is the only Candidate Key (CK)**.

2. Determine if R is in 2NF

- **Condition for 2NF:** A relation is in 2NF if it is in 1NF (which is assumed) and **every non-prime attribute is fully functionally dependent on the primary key**.
  - **Prime Attributes** (part of CK): P,Q,S.
  - **Non-Prime Attributes** (not part of CK): R,T.
  - **Partial Dependencies:** Check if a non-prime attribute depends on only a *proper subset* of the CK.
- **Check Non-Prime Attribute R:**
  - The FD is PQ→R.
  - PQ is a **proper subset** of the CK PQS.
  - Since R (non-prime) depends on PQ (a proper subset of the CK), this is a **partial dependency**.
- **Check Non-Prime Attribute T:**
  - The FD is S→T.
  - S is a **proper subset** of the CK PQS.
  - Since T (non-prime) depends on S (a proper subset of the CK), this is also a **partial dependency**.
- **Conclusion:** Since the relation has partial dependencies (PQ→R and S→T), **R is NOT in 2NF**.

GTU PREPZONE

3. Convert R to 2NF (Decomposition)

To convert to 2NF, we must remove the partial dependencies by decomposing the relation into smaller relations where the partial dependencies become full dependencies.

1. **Relation R1 (for PQ→R):**

    - This relation holds the attributes involved in the partial dependency.
    - R1 (P,Q,R) with CK: {P,Q} and FD: {PQ→R}. (Now R fully depends on PQ).
2. **Relation R2 (for S→T):**
    - This relation holds the attributes involved in the second partial dependency.
    - R2 (S,T) with CK: {S} and FD: {S→T}. (Now T fully depends on S).
3. **Relation R3 (Remaining):**
    - This relation holds the remaining attributes from the original CK that were not determinants in the partial dependencies. Here, no attributes are left, as P,Q,S were all determinants.
    - **Alternatively, a relation holding the full Candidate Key** PQS is needed to link the relations.
    - R3 (P,Q,S) with CK: {P,Q,S}.
    - *Correction based on decomposition principles: The attributes P,Q,S are already covered as keys in the decomposed relations. The primary key of the original relation should be kept intact in at least one decomposed relation to ensure lossless join.*

GTU PREPZONE

The 2NF Decomposition is:

- RA (P,Q,R) with CK {P,Q}
- RB (S,T) with CK {S}
- RC (P,Q,S) with CK {P,Q,S} (This relation is often necessary to preserve the dependency that links P,Q,S together, although in this case, PQS→All is already covered by the FDs.)

Simplified and Correct 2NF Decomposition (Minimal):

1. **R1 (P,Q,R)** (From PQ→R)
2. **R2 (S,T)** (From S→T)
3. **R3 (P,Q,S)** (The full CK to link the components)

All three new relations are in **2NF**.

Q.4 (a) Illustrate various storage strategies. (03 Marks)

Database storage strategies refer to how data is physically organized on disk to optimize access and efficiency.

1. **Heap File Organization:**

- Records are placed in any location where there is free space, in the order they are created.
    - **No specific order** is maintained, making sequential scans necessary for retrieval.
    - It is suitable for small relations or when access is primarily sequential.
2. **Sequential File Organization:**
    - Records are stored in a **sorted order** based on the values of a primary key or a specific sort key.
    - Highly efficient for sequential processing and range queries.
    - Insertion/deletion can be **inefficient** as it may require shifting records to maintain order.
3. **Hash File Organization:**
    - A **hash function** is used to calculate the address of the data record on the disk.
    - Provides **very fast direct access** (single record lookup) if the hash key is known.
    - Inefficient for range queries or sequential access.
4. **Clustering File Organization:**
    - Records from **different relations** that are frequently accessed together (often related by a one-to-many relationship) are stored physically close to each other.
    - This reduces the I/O cost for joins and linked data retrieval.
5. **Indexed Sequential Access Method (ISAM/B-Tree):**
    - Combines sequential organization with an **index** structure.
    - Allows both fast sequential access (using the file order) and fast indexed access (using the index).

Q.4 (b) Explain authorization and authentication with respect to database security. (04 Marks)

Database security involves protecting the data from unauthorized access, modification, or destruction. **Authentication** and **Authorization** are two fundamental pillars of this security.

1. Authentication (Who are you?)

- **Definition:** The process of **verifying the identity** of a user (or process) who is trying to access the database system.
- **Goal:** To confirm that the user is truly who they claim to be.

GTU PREPZONE

- **Mechanism:** Typically involves providing credentials like a **Username and Password**. Other methods include biometric scans, digital certificates, or multi-factor authentication.
- **Initial Step:** Authentication is the **first step** in the security process; a user must be authenticated before any operations can be performed.
- **Example:** A user logging into a DBMS by providing their DB_User and Password.

2. Authorization (What can you do?)

- **Definition:** The process of determining and granting the **specific permissions** or privileges a successfully authenticated user has on the database objects (tables, views, etc.).
- **Goal:** To control what actions an authenticated user is permitted to perform.
- **Mechanism:** Managed by the DBA using SQL commands like **GRANT** and **REVOKE** to assign permissions (e.g., SELECT, INSERT, UPDATE, DELETE).
- **Granularity:** Authorization can be set at various levels, such as the table level, column level, or even row level.
- **Example:** User 'Accountant' is **authorized** to SELECT and INSERT on the Employee table but is not permitted to DELETE.

GTU PREPZONE

Q.4 (c) Which kind of queries are solved using division operator? Explain in detail. (07 Marks)

The **Division Operator (÷)** in Relational Algebra is used to solve queries that involve the concept of "**for all**" or "**all-of-the-above**" conditions.

The Problem Solved by Division

1. **Query Type:** Division is typically used to find entities that are related to **all** entities in another set.
2. **Real-World Scenario:** It answers questions like:
   - "Find all **Students** who have taken **all** the courses offered by the 'CS' department."
   - "Find all **Projects** that use **all** the parts supplied by 'Supplier A'."
   - "Find all **Customers** who have purchased **every** product in the 'Electronics' category."
3. **Structure:** The division R÷S (where R and S are relations) finds the set of values from R (say, attribute X) that are associated with **every** value in S (attribute Y).

Explanation and Example

Let's use the query: **"Find the SupplierID of suppliers who supply *all* parts."**

- **Relation R (Supplier_Parts):** (Sno,Pno) — Records which supplier supplies which part.
- **Relation S (All_Parts):** (Pno) — Records the set of all existing parts.

The operation is R÷S (Project R onto Sno and Pno, then divide by S).

1. **Pre-requisite:** Let X be the set of attributes in R that are *not* in S (in the example, Sno). Let Y be the set of attributes in S (in the example, Pno).
2. **Definition of Division:** The result of R÷S is a relation containing all the values of X (the Sno) such that for every value of Y (the Pno) in S, the tuple (X,Y) exists in R.
3. **Result:** The resulting relation will be Sno, containing only those Sno values that are linked to every single Pno present in the All_Parts relation (S).
4. **Complexity:** While Division is conceptually simple, it is a complex operation that must be defined using the fundamental Relational Algebra operators (π,σ,×,∪,−) as follows:

$R \div S = \pi X \ (R) - \pi X \ (\pi X \ (R) \times S - R)$

GTU PREPZONE

OR Q.4 (a) Differentiate dynamic hashing and static hashing. (03 Marks)

Feature	Static Hashing	Dynamic Hashing
Directory Size	**Fixed** number of buckets/pages.	**Grows and shrinks** dynamically based on data.
Hash Function	A **fixed** hash function maps key to a bucket address.	A hash function is used, but the address space **changes** over time.
Data Growth	Difficult to handle. Leads to **overflow buckets** and performance degradation.	**Easily accommodates** data growth/shrinkage by splitting/merging buckets.
Performance	Degradation is high when file size fluctuates significantly.	**Maintains high performance** (low overflow) regardless of file

Feature	Static Hashing	Dynamic Hashing
		size.
Examples	Open addressing, Chaining.	**Extendable Hashing**, **Linear Hashing**.

Q.4 (b) Explain ACID properties of transaction. (04 Marks)

The **ACID** properties are a set of guarantees provided by a DBMS to ensure data integrity and reliability, even in the presence of system failures or concurrent transactions.

1. **Atomicity:**
   - **Concept:** A transaction must be treated as a single, **indivisible unit** of work.
   - **Rule:** Either all the operations of the transaction are executed successfully, or none of them are. If any part fails, the entire transaction is **rolled back** (undoing its effects).
   - **Mnemonic:** All or Nothing.
2. **Consistency:**
   - **Concept:** A transaction must take the database from one **valid state** to another valid state.
   - **Rule:** Any transaction, when executed in isolation, must preserve the database integrity constraints (like referential integrity, unique keys, etc.).
   - **Mnemonic:** Valid state to valid state.
3. **Isolation:**
   - **Concept:** Multiple concurrent transactions must be executed in a way that the result is the **same as if they were executed sequentially** (one after the other).
   - **Rule:** Each transaction must be unaware of any other transactions running concurrently; they must appear to run in isolation.
   - **Mnemonic:** Running alone.
4. **Durability:**
   - **Concept:** Once a transaction has been successfully **committed**, its changes to the database must be permanent, surviving subsequent system failures (e.g., power loss).
   - **Rule:** The committed changes are recorded in non-volatile storage (disk) and cannot be lost.
   - **Mnemonic:** Permanent changes.

GTU PREPZONE

Q.4 (c) Describe query processing with neat diagram. (07 Marks)

Query Processing is the sequence of activities and steps taken by a DBMS to execute a user's query and retrieve the requested data. Its goal is to execute the query as efficiently as possible (minimizing I/O costs and CPU time).

1. Parsing and Translation

- **Parser:** The SQL query is first checked for **syntactic and semantic correctness**.
- **Translation:** The valid SQL query is translated into a form that the DBMS can understand and manipulate, typically a Relational Algebra expression.

2. Optimization (Crucial Step)

- **Goal:** To find the **most efficient execution plan** from many logically equivalent plans.
- **Techniques:** The optimizer uses a **cost model** to estimate the cost (I/O, CPU) of different execution strategies (e.g., using indexes, join order).
- **Output:** The optimizer produces the optimal **Query Execution Plan** (a sequence of operations).

3. Evaluation

- **Query Execution:** The final stage where the database engine executes the chosen execution plan.
- **Operators:** The plan consists of low-level physical operators (e.g., nested-loop join, sequential scan, indexed lookup).
- **Result:** The operators are executed, and the final result of the query is returned to the user.

Q.5 (a) Explain working of two phase locking protocol. (03 Marks)

The **Two-Phase Locking (2PL)** Protocol is a concurrency control method that ensures **serializability** (a key part of Isolation) by regulating when a transaction can acquire and release locks.

1. **Principle:** Every transaction is required to request and release locks in two distinct phases.
2. **Growing Phase (Acquiring):**

GTU PREPZONE

    - A transaction can **obtain (acquire)** new locks (Shared or Exclusive) on the data items it needs.
    - It is **not allowed to release** any locks during this phase.
    - The point when the transaction acquires its final lock is called the **lock point**.
3. **Shrinking Phase (Releasing):**
    - A transaction can **release** its acquired locks.
    - It is **not allowed to obtain** any new locks during this phase.
    - Once a lock is released, the transaction cannot acquire any more locks.
4. **Guarantee:** 2PL guarantees that any schedule produced will be **serializable**.
5. **Drawback:** The basic 2PL protocol can potentially lead to a **deadlock** situation.

Q.5 (b) Explain GRANT, REVOKE and SAVEPOINT commands with suitable example. (04 Marks)

GTU PREPZONE

These are DCL (Data Control Language) and Transaction Control commands used in SQL.

1. GRANT Command

- **Purpose:** To assign **privileges** (permissions) to users or roles on specific database objects.
- **Syntax:** GRANT <privilege(s)> ON <object> TO <user/role>;
- **Privileges:** Common privileges include SELECT, INSERT, UPDATE, DELETE, REFERENCES.
- **Example:**

  SQL

  ```
  GRANT SELECT, INSERT ON Employees TO 'user_clerk';
  ```

- **Effect:** The user 'user_clerk' can now view and add records to the Employees table.

2. REVOKE Command

- **Purpose:** To **remove** privileges from users or roles that were previously granted.
- **Syntax:** REVOKE <privilege(s)> ON <object> FROM <user/role>;

- **Example:**

  SQL

  ```
  REVOKE DELETE ON Employees FROM 'user_clerk';
  ```

- **Effect:** The user 'user_clerk' can no longer delete records from the Employees table.

3. SAVEPOINT Command

- **Purpose:** To create a **named point** within a transaction to which the database can later be rolled back.
- **Context:** It is used within a larger transaction, before it is committed.
- **Syntax:** SAVEPOINT <savepoint_name>;
- **Example:**

  SQL

  ```
  INSERT INTO Accounts VALUES (1, 1000);
  SAVEPOINT before_transfer; -- Creates a rollback marker
  UPDATE Accounts SET Balance = Balance - 100 WHERE ID = 1;
  -- If a later operation fails, we can ROLLBACK TO before_transfer;
  ```

- **Effect:** It allows for partial rollbacks, reversing only the operations performed after the SAVEPOINT was established.

GTU PREPZONE

Q.5 (c) Assume table CUSTOMER (Cust_Id, Customer_name, Age, Address, Salary). Write a PL/SQL function which givens total number of customers having salary more than one lac per month. (07 Marks)

PL/SQL Function for Counting High-Salary Customers

```
CREATE OR REPLACE FUNCTION get_high_salary_customers

RETURN NUMBER

IS

    v_count NUMBER

BEGIN

    SELECT COUNT(*)
```

```
        INTO v_count
        FROM customers
        WHERE salary>100000;
        RETURN v_count
END;
/
```

- **To fetch data use:**

```
SEELCT get_high_salary_customers FROM dual;
```

OR Q.5 (a) Differentiate between conflict and view serializability with respect to transaction. (03 Marks)

Both **Conflict Serializability** and **View Serializability** are conditions used to determine if a concurrent schedule of transactions is equivalent to some serial schedule (i.e., if it guarantees Isolation).

GTU PREPZONE

Feature	Conflict Serializability (CS)	View Serializability (VS)
Definition	A schedule is CS if it can be transformed into a serial schedule by **swapping non-conflicting operations**.	A schedule is VS if it has the **same effect (view)** as some serial schedule.
Conflicting Ops	Involves two operations by different transactions on the same data item, where at least one is a **write** operation $(R-W, W-R, W-W)$.	Considers initial read, final write, and intermediate update reads (cascades).
Implementation	Generally, much **easier to test** (uses a precedence graph).	**NP-complete** (computationally expensive and difficult to test).
Guarantees	A stricter condition: all CS schedules are VS.	A more general condition: not all VS schedules are CS.
Use in DBMS	**Used in practice** by protocols	**Not used in practice** due to

Feature	Conflict Serializability (CS)	View Serializability (VS)
	like 2PL to enforce serializability.	testing complexity; primarily a theoretical concept.

Export to Sheets

OR Q.5 (b) Categorize joins in the SQL. Explain each with suitable example. (04 Marks)

SQL Joins are used to combine rows from two or more tables based on a related column between them.

1. **INNER JOIN:**
   - **Concept:** Returns only the rows that have **matching values** in both tables based on the join condition. Rows without a match are excluded.
   - **Example:**

     SQL

     ```
     SELECT E.Name, D.DeptName
     FROM Employee E INNER JOIN Department D
     ON E.DeptID = D.ID;
     ```

2. **LEFT (OUTER) JOIN:**
   - **Concept:** Returns **all rows from the left table** and the matching rows from the right table. If there is no match in the right table, NULLs are returned for the right table's columns.
   - **Example:**

     SQL

     ```
     SELECT C.Name, O.OrderID
     FROM Customer C LEFT JOIN Orders O
     ON C.ID = O.CustID;
     ```

3. **RIGHT (OUTER) JOIN:**
   - **Concept:** Returns **all rows from the right table** and the matching rows from the left table. If there is no match in the left table, NULLs are returned for the left table's columns.
   - **Example:**

GTU PREPZONE

```
SQL

SELECT P.Name, I.InStock
FROM Products P RIGHT JOIN Inventory I
ON P.ID = I.ProductID;
```

4. **FULL (OUTER) JOIN:**
   - **Concept:** Returns rows when there is a **match in one of the tables**. It returns all rows from both the left and right tables, placing NULLs where the match is missing.
   - **Example:**

```
SQL

SELECT A.City, B.Zip
FROM TableA A FULL JOIN TableB B
ON A.ID = B.ID;
```

5. **CROSS JOIN (Cartesian Product):**
   - **Concept:** Returns a result set that is the **Cartesian product** of the two tables. Every row from the first table is combined with every row from the second table.
   - **Example:** SELECT * FROM TableA CROSS JOIN TableB;

GTU PREPZONE

OR Q.5 (c) Write a PL/SQL trigger where employee of "GTU Private Ltd." company cannot update database on 23-Dec-2024 due to maintenance. (07 Marks)

PL/SQL Trigger for Maintenance Lockout

```
CREATE OR REPLACE TRIGGER prevent_update_on_maintenance

BEFORE UPDATE ON employees

FOR EACH ROW

    v_today DATE:=TRUNC(SYSDATE);

BEGIN

    IF v_today = TO_DATE('23-DEEC-2024','DD-MM-YYYY') THEN
```

```
                RAISE_APPLICATION_EERROR(-20001, 'database update are not
allowed on 23-DEC-2024 due to maintenance.')

        END IF;

END

/
```

GTU PREPZONE