**15 Batch**

**1 No.**

1. What is Database Management System(DBMS) and why is it important in modern computing?

### Database Management System (DBMS) (2 Marks)

A **Database Management System (DBMS)** is a software system that facilitates the creation, management, and manipulation of databases. It provides an interface for users and applications to interact with the database by allowing them to store, retrieve, update, and delete data in a structured and efficient way. A DBMS ensures that the data is organized in a manner that supports quick and reliable access, security, and consistency.

### Importance of DBMS in Modern Computing (2 Marks)

**Data Management:**

* 1. A DBMS allows efficient storage and retrieval of vast amounts of data, which is crucial in modern computing where large volumes of data are generated daily.

**Data Integrity and Consistency:**

* 1. DBMS enforces rules (integrity constraints) to ensure that the data remains accurate and consistent even in cases of system failure or concurrent access by multiple users.

**Security:**

* 1. A DBMS provides mechanisms to control access to sensitive data, ensuring that only authorized users can perform specific actions like reading, writing, or updating data.

**Data Redundancy Reduction:**

* 1. By storing data in a centralized system, a DBMS minimizes data duplication and ensures that the data is consistent across all applications using it.

**Support for Complex Queries:**

* 1. DBMS allows users to execute complex queries to retrieve and manipulate data efficiently using SQL, making it ideal for data analytics and decision-making processes in businesses and industries.

### Summary:

A DBMS is crucial in modern computing for organizing, managing, and securing large amounts of data while ensuring consistency, integrity, and accessibility. It plays a vital role in applications like banking systems, online retail, social networks, and data analytics.

1. 14 batch has answer  
     
   (C)There are different types of database-system users, differentiated by the way they expect to interact with the system. Explain each of them?

### Different Types of Database-System Users (4 Marks)

**Database Administrators (DBAs):**

* 1. **Role:** DBAs are responsible for managing and maintaining the database system. They perform tasks such as database design, security management, performance tuning, backup and recovery, and user access control.
  2. **Interaction:** DBAs interact with the system at an administrative level, ensuring its smooth operation and efficient management. They typically use tools for database management, performance monitoring, and security configuration.

**End Users:**

* 1. **Role:** End users are the individuals who directly interact with the database to perform specific tasks. They do not typically have technical knowledge about database management.
  2. **Interaction:** They interact with the system through application software, where they can query, update, and manipulate data in a user-friendly interface. For example, in a bank, an end user could be a customer checking their account balance via an app.

**Application Programmers (Developers):**

* 1. **Role:** Application programmers are responsible for designing and developing the software applications that interact with the database. They write the code that connects to the database, sends queries, and processes the retrieved data.
  2. **Interaction:** They use programming languages (e.g., Java, Python, C++) and query languages (e.g., SQL) to interact with the database system, creating functionalities that allow end users to work with data.

**System Analysts:**

* 1. **Role:** System analysts design and implement the overall structure of the database system by analyzing user requirements. They work closely with DBAs and developers to ensure that the database system meets the needs of the organization.
  2. **Interaction:** Analysts typically create detailed database specifications, including defining data models and ensuring the system’s design meets both business and technical requirements.

### Summary:

* **DBAs** manage and maintain the database.
* **End users** interact with the database through applications.
* **Application programmers** develop applications that connect to the database.
* **System analysts** design the system to meet user needs and specifications.

**2 No.  
  
(A)** List two reasons why null value may be introduced in database?

### Two Reasons Why Null Values May Be Introduced in a Database (2 Marks)

**Missing Information:**  
A value might be unknown at the time of data entry. For example, a customer's phone number might not be available when their record is first created.

**Not Applicable Information:**  
Certain attributes may not apply to all records. For example, a "Spouse Name" field would be null for an unmarried person in an employee database.

### Summary:

Null values help handle incomplete or irrelevant data, maintaining database flexibility and integrity.

1. Discuss the relative merits of procedural and non-procedural language?

### Relative Merits of Procedural and Non-Procedural Languages (2 Marks)

**1. Procedural Language:**

* **Definition:** Specifies the steps or procedures to be followed to achieve a task.
* **Merits:**
  + Provides detailed control over data manipulation and execution flow.
  + Easier to optimize for performance, as developers specify exact operations.
* **Example:** SQL's procedural extensions (PL/SQL), C++, Java.

**2. Non-Procedural Language:**

* **Definition:** Describes what the task is, rather than how to perform it.
* **Merits:**
  + Simpler for users; focuses on specifying results rather than steps.
  + Often more concise and easier to learn, reducing development time.
* **Example:** Standard SQL, Python's data query libraries.

### Summary:

* **Procedural languages** offer control and flexibility but require detailed coding.
* **Non-procedural languages** are easier to use and understand, making them suitable for users focusing on data outcomes rather than logic flow.

1. Consider the SQL query   
   select distinct p.a1  
   from p,r1,r2  
   where p.a1=r1,a1 or p.a1=r2.a1  
   Under what condition does the preceding query select values of p.a1 that are either in r1 or in r2? Examine carefully the case where one of r1 or r2 may be empty?

### SQL Query Analysis:

Given the query:

SELECT DISTINCT p.a1

FROM p, r1, r2

WHERE p.a1 = r1.a1 OR p.a1 = r2.a1;

### Explanation:

The query retrieves distinct values of p.a1 that satisfy either of the following conditions:

1. p.a1 exists in the a1 column of table r1, **OR**
2. p.a1 exists in the a1 column of table r2.

### Condition for Selecting Values:

* The query will return a value of p.a1 if it matches any value in either r1.a1 or r2.a1.
* If a match is found in either table, the row is included in the result. The DISTINCT keyword ensures that duplicates are removed from the final output.

### Special Case: When One Table Is Empty:

**If** r1 **is empty:**

* + The condition p.a1 = r1.a1 is never true.
  + The query behaves as if it only checks the condition p.a1 = r2.a1, so it returns values of p.a1 that exist in r2.

**If** r2 **is empty:**

* + The condition p.a1 = r2.a1 is never true.
  + The query behaves as if it only checks the condition p.a1 = r1.a1, so it returns values of p.a1 that exist in r1.

**If both** r1 **and** r2 **are empty:**

* + The query returns an empty result set since neither condition is true for any value of p.a1.

### Conclusion:

The query selects values of p.a1 that exist in either r1.a1 or r2.a1. If one of the tables is empty, the query still works, but it only checks the condition involving the non-empty table. If both are empty, no results are returned.

(D) Show that, in SQL, <> all is identical to not in?

### Demonstrating <> ALL is Identical to NOT IN in SQL (3 Marks)

**1. Understanding** <> ALL**:**  
The expression x <> ALL (subquery) evaluates to TRUE if x is not equal to **any** value returned by the subquery. It means x must be different from **all** values in the set.

**2. Understanding** NOT IN**:**  
The expression x NOT IN (subquery) evaluates to TRUE if x does not match **any** value returned by the subquery. It also means x must be different from **all** values in the set.

**3. Equivalence Demonstration:**

Consider the following example:

SELECT \* FROM customers

WHERE customer\_id <> ALL (SELECT customer\_id FROM orders);

This query checks if customer\_id is different from all customer\_id values in the orders table.

The equivalent NOT IN query is:

SELECT \* FROM customers

WHERE customer\_id NOT IN (SELECT customer\_id FROM orders);

**Result:**  
Both queries return customers whose customer\_id is not found in the orders table. They behave identically.

### Conclusion:

* <> ALL and NOT IN both ensure that a value is different from all values in a subquery result.
* Therefore, they are functionally equivalent in SQL.

**3 No.  
(A)** List some of the common data types supported by SQL?

### Common Data Types Supported by SQL (3 Marks)

**Numeric Data Types:**

* 1. **INT (INTEGER):** Whole numbers (e.g., 100, -45).
  2. **FLOAT/REAL:** Floating-point numbers for approximate decimal values (e.g., 12.34).
  3. **DECIMAL (NUMERIC):** Exact numeric values with fixed precision (e.g., 123.45).

**Character Data Types:**

* 1. **CHAR(n):** Fixed-length string of n characters (e.g., CHAR(10)).
  2. **VARCHAR(n):** Variable-length string up to n characters (e.g., VARCHAR(50)).

**Date and Time Data Types:**

* 1. **DATE:** Stores dates (e.g., 2024-01-01).
  2. **TIME:** Stores time (e.g., 12:30:45).
  3. **TIMESTAMP:** Stores date and time together (e.g., 2024-01-01 12:30:45).

### Summary:

SQL provides various data types for handling **numeric**, **character**, and **date/time** data, ensuring efficient data storage and management.

1. Consider the bank database. Give an expression in the relational algebra for each of the following queries:  
     
   branch(branch\_name, branch city, assets)  
   customer(customer\_name, customer street, customer city)  
   loan(loan\_number, branch name, amount)  
   borrower(customer\_name, loan number)  
   account(account number, branch name, balance)  
   depositor(customer\_name, account number)  
     
   (i) Write an SQL query to find the name of all customers whose balance is over 900000 and whose branch name is Trishal.  
   (ii) Write an SQL query to find all customer who have a loan but no an account at the bank.

### ****(i) Find the names of all customers whose balance is over 900,000 and whose branch name is 'Trishal':****

SELECT DISTINCT d.customer\_name

FROM depositor d

JOIN account a ON d.account\_number = a.account\_number

WHERE a.balance > 900000AND a.branch\_name = 'Trishal';

**Explanation:**

* This query joins the depositor and account tables based on the account\_number.
* It filters accounts with a balance greater than 900,000 and belonging to the 'Trishal' branch.
* DISTINCT ensures that duplicate customer names are removed from the result.

### ****(ii) Find all customers who have a loan but no account at the bank:****

SELECT DISTINCT b.customer\_name

FROM borrower b

WHERE b.customer\_name NOT IN (

SELECT d.customer\_name

FROM depositor d

);

**Explanation:**

* This query selects customers from the borrower table who do not appear in the depositor table.
* The subquery retrieves all customer names with accounts.
* NOT IN ensures only those borrowers who do not have an account are returned.

1. In what ways data mining and data warehousing are closely connected to DBMS?

### How Data Mining and Data Warehousing are Connected to DBMS (3 Marks)

\*\*1. **Data Warehousing and DBMS:**

* A **data warehouse** stores large volumes of historical data, aggregated from different sources, which is managed using a DBMS.
* The DBMS provides data integration, storage, and querying capabilities, ensuring data consistency and accessibility.

**2. Data Mining and DBMS:**

* **Data mining** involves extracting patterns and knowledge from large datasets, which are often stored in a DBMS or data warehouse.
* DBMS supports data mining by providing efficient storage, retrieval, and processing of large datasets for analysis.

**3. Interconnection:**

* Data warehouses rely on DBMS to manage and organize data.
* Data mining uses the structured data from warehouses managed by DBMS to identify trends and insights.

### Conclusion:

Data warehousing structures and stores data using DBMS, while data mining extracts meaningful information from this data, showcasing their interconnected roles in data management.

4 No.  
  
(A) What are the different types of attribute in the ER model? Provide examples for each type.

### Types of Attributes in the ER Model (4 Marks)

In the Entity-Relationship (ER) model, attributes represent the properties of an entity or relationship. The different types of attributes include:

**1. Simple (Atomic) Attribute:**

* **Definition:** Cannot be divided further into smaller parts.
* **Example:**
  + Employee entity with attribute employee\_id or age.

**2. Composite Attribute:**

* **Definition:** Can be divided into smaller sub-parts, each representing a more basic attribute.
* **Example:**
  + Full Name can be split into first\_name, middle\_name, and last\_name.
  + Address into street, city, state, and zip\_code.

**3. Derived Attribute:**

* **Definition:** Value can be derived from other attributes in the database.
* **Example:**
  + Age derived from date\_of\_birth.
  + Total Salary derived from basic\_salary and bonus.

**4. Multi-Valued Attribute:**

* **Definition:** Can hold multiple values for a single entity instance.
* **Example:**
  + An Employee entity having multiple phone\_numbers or email\_addresses.

**5. Key Attribute:**

* **Definition:** Uniquely identifies an entity instance within an entity set.
* **Example:**
  + student\_id in a Student entity.
  + ISBN in a Book entity.

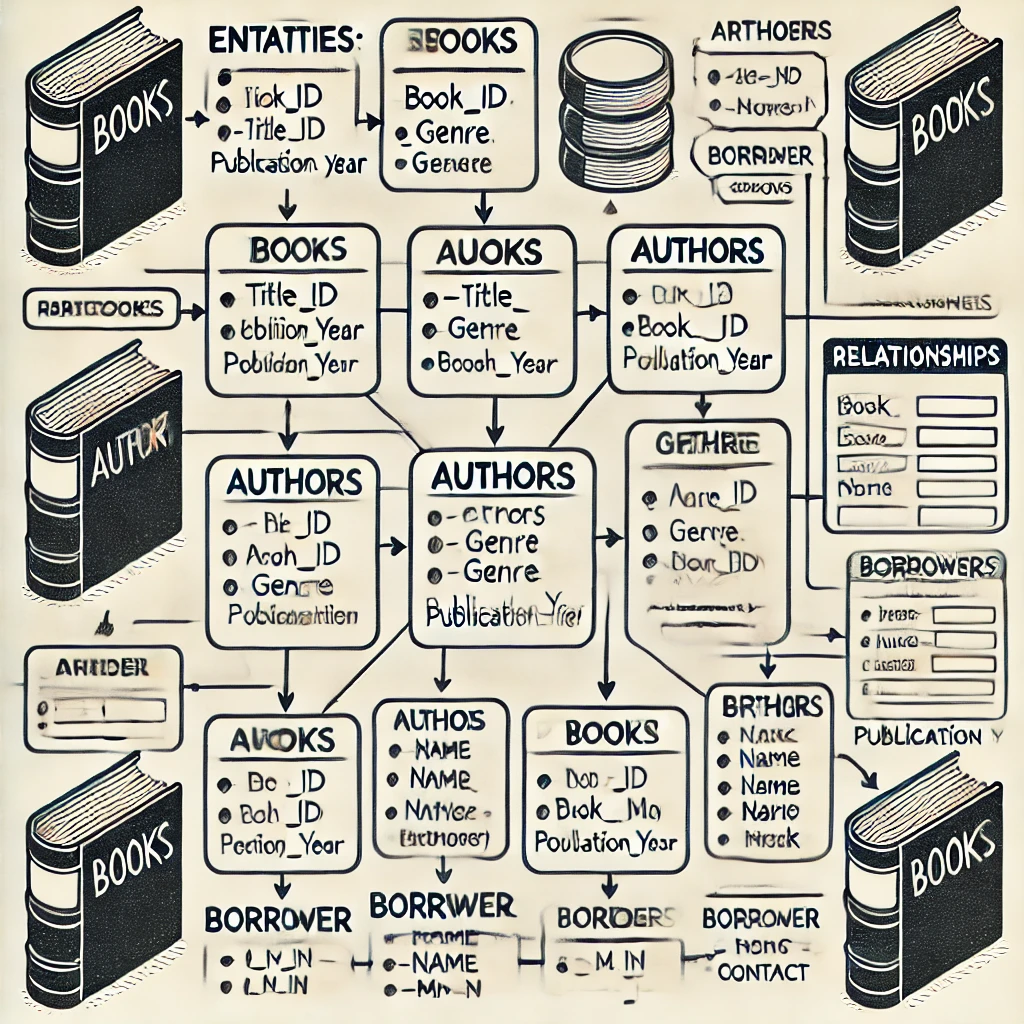
**6. Null Attribute:**

* **Definition:** Represents the absence of a value.
* **Example:**
  + spouse\_name for an unmarried Employee.

### Conclusion:

The ER model classifies attributes into types like simple, composite, derived, multi-valued, and key attributes, each serving a distinct role in defining entity properties.

1. Draw an ER diagram for a simple library database that includes entites for books, authors and borrowers. Include relationships and cardinalities.



1. Explain what a weak entity is and how it differs from a strong entity?

### Weak Entity vs Strong Entity (3 Marks)

**1. Weak Entity:**

* **Definition:** A weak entity is an entity that cannot be uniquely identified by its own attributes alone. It depends on a **strong (or owner) entity** for its identification. It requires a **partial key** (a set of attributes that can identify an entity within the weak entity set) along with a reference to the strong entity.
* **Example:** Consider an entity Order\_Item in a library database, where each order item is associated with a specific order. The Order\_Item entity may require the Order\_ID (from the Order entity) and a partial key like Item\_Number to uniquely identify each item.

**2. Strong Entity:**

* **Definition:** A strong entity is an entity that can be uniquely identified by its own attributes, without relying on any other entity. It has a **primary key** that uniquely identifies each instance of the entity.
* **Example:** An Author entity, where Author\_ID can uniquely identify each author.

### Key Differences:

| **Aspect** | **Weak Entity** | **Strong Entity** |
| --- | --- | --- |
| **Identification** | Cannot be uniquely identified by its own attributes. Relies on a strong entity. | Can be uniquely identified by its own attributes. |
| **Key** | Has a partial key (combined with a strong entity). | Has a primary key that uniquely identifies it. |
| **Existence** | Cannot exist without the associated strong entity. | Can exist independently. |

5 No.  
  
(A) What are the significance's of triggers in DBMS? Write an example of trigger with proper syntax?

### Significance of Triggers in DBMS (4 Marks)

**1. Automation of Actions:**  
Triggers automate certain actions within a database, such as updating records, deleting records, or enforcing business rules, without requiring explicit calls to perform those actions. This reduces the chances of human error and ensures that the database behaves as expected.

**2. Enforcing Data Integrity and Consistency:**  
Triggers help maintain data integrity by automatically checking conditions and enforcing rules (like preventing invalid data from being entered) before an action is performed. For example, a trigger can prevent an update to a record if certain conditions are not met.

**3. Auditing and Monitoring Changes:**  
Triggers are commonly used for auditing purposes to record any changes made to the database. For example, when a record is updated or deleted, a trigger can log the change in an audit table for future reference.

**4. Preventing Invalid Transactions:**  
Triggers can be used to prevent certain transactions that violate business rules, ensuring the consistency of the database. For example, a trigger can be used to prevent the deletion of a record if it is linked to other records in the database.

### Example of a Trigger:

Let's consider a scenario where we want to automatically log the changes made to the employees table. Whenever an update is made to the salary field, the change is recorded in an audit\_log table.

**Trigger Syntax:**

CREATE TRIGGER salary\_update\_trigger

AFTER UPDATE ON employees

FOR EACH ROWBEGIN

IF OLD.salary != NEW.salary THEN

INSERT INTO audit\_log (employee\_id, old\_salary, new\_salary, change\_date)

VALUES (NEW.employee\_id, OLD.salary, NEW.salary, NOW());

END IF;

END;

### Explanation of the Trigger:

* **Trigger Name:** salary\_update\_trigger
* **Timing:** AFTER UPDATE — This trigger fires after an update operation on the employees table.
* **Condition:** The trigger only executes if the salary has changed (OLD.salary != NEW.salary).
* **Action:** When the salary changes, a record is inserted into the audit\_log table, storing the employee's ID, the old and new salary, and the timestamp of the change.

### Conclusion:

Triggers in DBMS are powerful tools that automate actions, enforce rules, ensure data consistency, and facilitate auditing, making them a crucial part of modern database management.

1. Define the concept of roles in the context of database privileges. Give an example of it.

### Concept of Roles in the Context of Database Privileges (4 Marks)

In the context of database privileges, **roles** are a way to group multiple privileges together and assign them to users or other roles. Roles help in managing user access and permissions in a more organized and efficient way. Instead of granting individual privileges to each user, a role can be created that encapsulates a set of privileges, and then that role is assigned to users. This simplifies the management of permissions, especially in large organizations with many users.

### Key Points:

1. **Role-Based Access Control (RBAC):** Roles are used in RBAC, where each user is assigned one or more roles, and each role has specific privileges. This ensures that users can access only the data and perform actions that are relevant to their role within the organization.
2. **Privileges:** Privileges refer to the specific rights (such as SELECT, INSERT, UPDATE, DELETE) granted on database objects (such as tables, views, or procedures).
3. **Role Assignment:** Users are assigned roles based on their job responsibilities, making privilege management more flexible and easier to audit.

### Example of Role and Privileges:

Let’s say we have a database with two types of users: **Managers** and **Employees**. Managers need full access to all tables in the database, while Employees should only have read access to the data.

1. **Create Roles:**

-- Create a role for managers with full privileges

CREATE ROLE manager\_role;

-- Create a role for employees with read-only access

CREATE ROLE employee\_role;

1. **Assign Privileges to Roles:**

-- Grant privileges to the manager\_role

GRANT SELECT, INSERT, UPDATE, DELETE ON employees TO manager\_role;

-- Grant only SELECT (read-only) privileges to the employee\_role

GRANT SELECT ON employees TO employee\_role;

1. **Assign Roles to Users:**

-- Assign the manager\_role to a user named 'john'

GRANT manager\_role TO john;

-- Assign the employee\_role to a user named 'mary'

GRANT employee\_role TO mary;

### Explanation:

* **Manager Role:** The manager\_role is granted full privileges (SELECT, INSERT, UPDATE, DELETE) on the employees table. Any user assigned to this role, such as john, will have these permissions.
* **Employee Role:** The employee\_role is granted only the SELECT privilege on the employees table. Users assigned to this role, such as mary, can only view the data but cannot modify it.

### Conclusion:

Roles in database management systems are essential for grouping and managing user privileges efficiently, ensuring proper access control while reducing the complexity of assigning permissions to individual users.

1. Explain how the concept of privileges contributes to the overall security of a database

### Contribution of Privileges to Database Security (4 Marks)

**1. Access Control:**  
Privileges define who can access specific database objects and what actions they can perform (e.g., SELECT, INSERT, UPDATE, DELETE). By carefully assigning privileges, administrators can ensure that only authorized users can interact with sensitive data, protecting it from unauthorized access.

**Example:** A user might be granted the SELECT privilege on a customer table, but not the INSERT or DELETE privileges, ensuring that the user can only view the data, not modify it.

**2. Principle of Least Privilege (PoLP):**  
Privileges enable the enforcement of the Principle of Least Privilege, which ensures that users only have the minimum permissions necessary to perform their job functions. This reduces the risk of accidental or malicious data modifications.

**Example:** A junior employee might only need read-only access to customer records, so the DBA grants only the SELECT privilege, preventing any unauthorized data manipulation.

**3. Fine-Grained Access Control:**  
Privileges allow for **granular control** over who can access specific parts of the database, such as individual columns or rows, not just entire tables. This helps protect sensitive data within the database.

**Example:** A user may have access to most columns of the employee table but be restricted from viewing the salary column, which contains sensitive information.

**4. Auditing and Monitoring:**  
Privileges help track and monitor the actions of users. By assigning different levels of access, administrators can monitor user activities more effectively, ensuring that any suspicious or unauthorized actions are detected and addressed promptly.

**Example:** If a user with elevated privileges tries to access restricted data or perform unauthorized operations, the database can log the action for auditing purposes.

### Conclusion:

In summary, privileges are essential to database security as they control access, enforce data integrity, minimize the risk of unauthorized actions, and enable detailed monitoring and auditing of database activities. Proper privilege management ensures that sensitive data is protected while allowing legitimate users to perform their tasks.

**6 No.  
(A)**  Consider the following University Database  
 department(dept\_name, building, budget)

Course(course\_id,title,dept\_name,credits)  
 section(course\_id,sec\_id,semester,year,building,

room\_number, time\_slot)

teaches(ID, course\_id, sec\_id, semester,year)  
Write the SQL queries for each of the following condition:  
(i) Find the branches where the average account balance is more than TK. 3000. Find the names of all instructors who have a higher salary than some instructor in “Comp.Sci”  
(ii) Find the names of all instructors in the university who have taught some courses, find the name and course\_id

#### (i) Find the branches where the average account balance is more than TK. 3000. Find the names of all instructors who have a higher salary than some instructor in “Comp.Sci”.

**1. Query to find branches where the average account balance is more than TK. 3000:** Assuming you meant "branches" as a metaphor for departments and "account balance" as the budget of departments, this query will find the departments (branches) where the average budget is more than TK. 3000.

SELECT dept\_name

FROM department

GROUP BY dept\_nameHAVING AVG(budget) > 3000;

**Explanation:**

* The AVG(budget) function calculates the average budget for each department.
* The HAVING clause filters the departments with an average budget greater than 3000.

**2. Query to find the names of all instructors who have a higher salary than some instructor in "Comp.Sci":**

The assumption here is that the teaches table stores the instructor's ID, but we don't have a salary column in the given schema. To find instructors who have a higher salary than some instructor in "Comp.Sci", you would need a salary column in a table like instructor.

However, using the existing schema, we can still write a query to find instructors in a department. Here's an example that finds instructors teaching in the "Comp.Sci" department.

SELECT DISTINCT t1.IDFROM teaches t1WHERE EXISTS (

SELECT 1

FROM teaches t2

JOIN course c ON t2.course\_id = c.course\_id

WHERE c.dept\_name = 'Comp.Sci'

AND t1.ID != t2.ID

);

**Explanation:**

* This query assumes that an instructor teaching in "Comp.Sci" may have taught some course. The outer query selects instructors teaching a course, and the EXISTS clause checks if there is any other instructor in the "Comp.Sci" department.

#### (ii) Find the names of all instructors in the university who have taught some courses. Find the name and course\_id.

To list all instructors who have taught courses and the corresponding course names and IDs:

SELECT DISTINCT i.ID, c.title, c.course\_id

FROM teaches t

JOIN course c ON t.course\_id = c.course\_id

JOIN instructor i ON t.ID = i.ID;

**Explanation:**

* The query joins the teaches table with the course table on the course\_id, then joins the instructor table on the instructor's ID.
* It retrieves the distinct ID of the instructor, the title of the course, and the course\_id for each course the instructor has taught.

### Conclusion:

These queries address the required conditions based on the provided schema and assumptions. You may need to adjust the schema or tables to accurately represent data like instructor salaries, but the queries above are based on the available information.

1. Explain how integrity constraints guard against accidental damage to the database?

### How Integrity Constraints Guard Against Accidental Damage to the Database (3 Marks)

Integrity constraints are essential mechanisms in a database that help maintain the accuracy and consistency of the data. They protect the database from accidental damage by enforcing rules that prevent invalid, inconsistent, or corrupted data from being entered or modified. Here's how they work:

**Entity Integrity**:

* 1. **Prevents Duplicate or Null Primary Keys:** By enforcing primary key constraints, the database ensures that each record in a table is unique and identifiable. This prevents accidental insertion of duplicate rows or missing values in critical fields (such as primary keys).
  2. **Example:** If a student\_id field is defined as the primary key, it cannot contain null values or duplicates, ensuring each student has a unique identifier.

**Referential Integrity**:

* 1. **Prevents Orphaned Records:** Through foreign key constraints, referential integrity ensures that relationships between tables are maintained. It prevents actions like deleting records that are referenced by other tables, which could cause orphaned data.
  2. **Example:** In a orders table that references customer\_id in the customers table, the foreign key constraint prevents the deletion of a customer record if that customer has any associated orders.

**Check Constraints**:

* 1. **Prevents Invalid Data Entry:** Check constraints restrict the values that can be inserted into a column, ensuring only valid data is entered. This helps prevent errors like entering negative numbers in a field that should only have positive values.
  2. **Example:** A check constraint on the age column (CHECK (age >= 0)) prevents accidental entry of invalid or inappropriate data.

### Conclusion:

By enforcing entity integrity, referential integrity, and data validation rules (like check constraints), integrity constraints guard the database from accidental data corruption, ensuring that the data remains consistent, accurate, and reliable.

1. How SQL aggregation handles queries on relation containing null values?

### How SQL Aggregation Handles Queries on Relations Containing Null Values (3 Marks)

SQL aggregation functions like COUNT(), SUM(), AVG(), MIN(), and MAX() are commonly used to compute summary statistics on columns. When dealing with relations containing **NULL values**, SQL treats these values in specific ways depending on the aggregation function. Here's how each function handles NULL values:

**COUNT()**:

* 1. **Behavior:** The COUNT() function counts **only non-NULL values** in a column.
  2. **Example:** If a column salary has 10 rows, but 2 of them contain NULL values, COUNT(salary) will return 8 (ignoring the NULLs).
  3. **SQL Query:** SELECT COUNT(salary) FROM employees;

**SUM()**:

* 1. **Behavior:** The SUM() function adds up the values of a column, but it **ignores NULL values**.
  2. **Example:** If a column sales\_amount has the values (100, NULL, 200), SUM(sales\_amount) will return 300 (NULL is ignored in the summation).
  3. **SQL Query:** SELECT SUM(sales\_amount) FROM sales;

**AVG()**:

* 1. **Behavior:** The AVG() function calculates the average of non-NULL values in a column. It first sums the non-NULL values and then divides by the count of non-NULL entries.
  2. **Example:** If a column score has the values (10, 20, NULL), AVG(score) will return 15, calculated as (10 + 20) / 2.
  3. **SQL Query:** SELECT AVG(score) FROM results;

**MIN() and MAX()**:

* 1. **Behavior:** Both MIN() and MAX() functions ignore NULL values when determining the minimum or maximum of a column.
  2. **Example:** If a column age has the values (25, NULL, 30), MIN(age) will return 25 and MAX(age) will return 30, ignoring the NULL value.
  3. **SQL Query for MIN:** SELECT MIN(age) FROM employees;
  4. **SQL Query for MAX:** SELECT MAX(age) FROM employees;

### Conclusion:

SQL aggregation functions typically **ignore NULL values** when performing calculations (except for COUNT(), which only counts non-NULL values). As a result, it's important to account for NULLs in your queries to avoid any unintended data omissions or miscalculations.

**13 Batch**

**1 No.**

1. What do understand by DBMS? What kind of problems do we face to deal with data if we do not have database management system?

### What is DBMS? (4 Marks)

A **Database Management System (DBMS)** is software that provides an interface to interact with databases. It allows users to define, manipulate, retrieve, and manage data efficiently and securely. The DBMS provides functionalities such as data storage, retrieval, updating, and security management while ensuring data integrity, consistency, and concurrency control.

### Key Functions of DBMS:

1. **Data Definition:** The DBMS helps define the structure of data, using schemas and tables.
2. **Data Manipulation:** It allows for inserting, updating, deleting, and querying data.
3. **Data Security:** Ensures only authorized users can access or modify data.
4. **Concurrency Control:** Manages simultaneous data access by multiple users to prevent conflicts.
5. **Backup and Recovery:** Provides mechanisms to back up data and restore it in case of failure.

### Problems Without a DBMS

Without a DBMS, managing data manually or using file-based systems can lead to several problems:

**Data Redundancy:**

* 1. **Problem:** Data might be duplicated across multiple files or applications, leading to inconsistencies and inefficiencies.
  2. **Example:** Storing customer information in several departments (sales, inventory, accounts) can lead to repeated data and increased storage costs.

**Data Inconsistency:**

* 1. **Problem:** Without centralized control, there is no guarantee that the same data is consistent across different places. If one copy of the data is updated and others aren't, inconsistencies arise.
  2. **Example:** If the customer's address is updated in one system but not in others, conflicting information may result.

**Difficulty in Data Access:**

* 1. **Problem:** Without a DBMS, querying the data manually from multiple files becomes complex and error-prone.
  2. **Example:** Searching for a specific customer's order history across different files can be time-consuming and cumbersome without a unified system.

**Data Security and Integrity Issues:**

* 1. **Problem:** Ensuring data security, integrity, and authorization becomes much more difficult without a centralized DBMS system. Without proper controls, unauthorized access to data may occur.
  2. **Example:** Without a DBMS, there's no way to enforce roles and privileges for different users, leading to potential misuse of data.

**Lack of Backup and Recovery:**

* 1. **Problem:** Data loss can occur easily without backup and recovery mechanisms that are built into a DBMS.
  2. **Example:** If a file system crashes, manually recovering data without a structured database management system can be difficult and lead to loss of important data.

1. Write and explain the five main responsibilities of database administrator?

### Five Main Responsibilities of a Database Administrator (DBA) (4 Marks)

A **Database Administrator (DBA)** is responsible for managing and maintaining the database systems. The five key responsibilities are:

**Database Design**:

* 1. The DBA designs the structure of the database, including tables, relationships, and keys. This ensures that the database meets the business requirements efficiently.

**Security Management**:

* 1. The DBA sets up user roles, permissions, and access control to ensure that only authorized users can access sensitive data.

**Backup and Recovery**:

* 1. The DBA is responsible for creating regular backups and ensuring that data can be restored in case of a failure, thus preventing data loss.

**Performance Monitoring**:

* 1. The DBA monitors database performance, optimizes queries, and ensures efficient use of resources to keep the system running smoothly.

**Database Maintenance**:

* 1. The DBA handles routine maintenance tasks like applying updates, patches, and ensuring the system is running on the latest version to avoid security risks.

These responsibilities ensure the database operates securely, efficiently, and reliably.

**2 No.  
(A)** Consider the following University Database department(dept\_name, building, budget) Course(course\_id,title,dept\_name,credits) section(course\_id,sec\_id,semester,year,building, room\_number, time\_slot) teaches(ID, course\_id, sec\_id, semester,year)

Give an expression in SQL for each of the following queries:  
(i) Find the average salary of the instructors in each department.  
(ii) To find the name of all instructors in the university who have taught some course, find their names and course id.  
(iii) Find the names of all departments whose building name includes the substring ‘Watson’

### (i) Find the average salary of the instructors in each department.

To calculate the average salary for instructors in each department, we assume that the teaches table has the instructor's ID, and the ID field matches the instructor's ID in a instructor table (if salary data is stored there). If salary is stored in the instructor table, the query would look like this:

SELECT d.dept\_name, AVG(i.salary) AS average\_salary

FROM department d

JOIN course c ON d.dept\_name = c.dept\_name

JOIN teaches t ON c.course\_id = t.course\_id

JOIN instructor i ON t.ID = i.I

DGROUP BY d.dept\_name;

* **Explanation**: This query joins the department, course, teaches, and instructor tables to calculate the average salary of instructors (AVG(i.salary)) in each department.

### (ii) To find the name of all instructors in the university who have taught some course, find their names and course id.

This query assumes the instructor table has an instructor\_name field:

SELECT DISTINCT i.instructor\_name, t.course\_id

FROM instructor I

JOIN teaches t ON i.ID = t.ID;

* **Explanation**: This query selects distinct pairs of instructor names and the corresponding course IDs from the instructor and teaches tables, showing instructors who have taught courses.

### (iii) Find the names of all departments whose building name includes the substring ‘Watson’.

To search for departments whose building name includes the substring "Watson", we can use the LIKE operator in SQL:

SELECT dept\_name

FROM department

WHERE building LIKE '%Watson%';

* **Explanation**: This query retrieves the names of all departments where the building name contains the substring "Watson" using the LIKE operator with wildcard %.

These queries address the given tasks by leveraging SQL joins, aggregate functions, and string matching to retrieve the necessary information from the database.

1. Which level of abstraction describe about what data are stored in the database, and what relationship exist among those data?

The level of abstraction that describes **what data are stored in the database** and **what relationships exist among those data** is called the **Conceptual Level** or **Conceptual Schema**.

### Explanation:

* **Conceptual Level**: This level provides a high-level view of the entire database and focuses on the **structure** of the data, including what data are stored (e.g., tables, entities) and how different data elements are related (e.g., relationships, foreign keys). It abstracts away physical storage details and is concerned with **what data are represented and how they are logically related**.

### Key Characteristics of the Conceptual Level:

* Describes the logical structure of the database, such as entities, attributes, and relationships.
* Independent of physical storage and implementation details.
* Typically represented by an **Entity-Relationship (ER) model** or **Relational model** in database design.

In contrast:

* The **Physical Level** describes **how** the data are stored (e.g., file structures, indexes).
* The **External Level** describes how users or applications view the data (e.g., views and user interfaces).

**3 No.**

1. Done
2. branch(branch\_name, branch city, assets)  
   customer(customer\_name, customer street, customer city)  
   loan(loan\_number, branch name, amount)  
   borrower(customer\_name, loan number)  
   account(account number, branch name, balance)  
   depositor(customer\_name, account number)  
   (i) Write an SQL statement to insert a tuple in account relation where the account number is “A-1001” branch name is Mymensingh branch and balance is 1000.  
   (ii) Write an SQL query to find all customer who have both and an account at the bank

### (i) Insert a tuple into the account relation

To insert a new record into the account table where the account number is "A-1001", the branch name is "Mymensingh branch", and the balance is 1000, you would use the following INSERT statement:

INSERT INTO account (account\_number, branch\_name, balance)

VALUES ('A-1001', 'Mymensingh branch', 1000);

* **Explanation**: This query inserts a new row into the account table with the specified account number, branch name, and balance.

### (ii) Find all customers who have both a loan and an account at the bank

To find customers who have both a loan and an account, you need to perform a JOIN between the borrower and depositor tables, as these tables link customers with their loans and accounts, respectively. The SQL query would be:

SELECT DISTINCT b.customer\_name

FROM borrower b

JOIN depositor d ON b.customer\_name = d.customer\_name;

* **Explanation**: This query joins the borrower table (which connects customers to loans) with the depositor table (which connects customers to accounts) based on the customer\_name. The DISTINCT keyword ensures that each customer who has both a loan and an account is listed only once.

1. What do you mean by data mining and data warehousing ?

### ****Data Mining:****

Data mining is the process of discovering hidden patterns, trends, and relationships in large sets of data using statistical, mathematical, and computational techniques. It helps in making data-driven decisions by identifying patterns that can predict future trends or behaviors.

### ****Data Warehousing:****

Data warehousing is the practice of collecting, storing, and managing large volumes of data from different sources into a centralized repository. It is optimized for reporting and data analysis, making it easier for businesses to perform complex queries and generate insights for decision-making.

Together, data warehousing provides the infrastructure for storing data, and data mining helps to extract valuable knowledge from that data.

### ****Relationship between Data Mining and Data Warehousing:****

* Data **warehousing** provides a structured environment for storing large volumes of data from multiple sources.
* **Data mining** techniques are then used on this data in the warehouse to uncover useful patterns, trends, and insights.

In short, data mining is the process of analyzing the data, while data warehousing is about storing and organizing that data.

**5 No.   
(A)** Explain how integrity constraints guard against accidental damage to the database.

### ****How Integrity Constraints Guard Against Accidental Damage to the Database (4 Marks)****

**Integrity constraints** are rules defined on a database to ensure the accuracy, consistency, and reliability of the data. These constraints prevent the accidental insertion, deletion, or modification of incorrect or inconsistent data, thus safeguarding the database from errors. Here's how they help:

**Entity Integrity**:

* 1. This ensures that each record in a table is uniquely identifiable by a primary key. The primary key cannot be null, ensuring no duplicate records in the table.
  2. **Example**: If a customer’s ID is used as a primary key, this constraint ensures no two customers can have the same ID, protecting the uniqueness of customer data.

**Referential Integrity**:

* 1. This constraint ensures that relationships between tables are maintained by enforcing valid foreign key references. It prevents data inconsistencies, such as having a foreign key value that doesn't exist in the referenced table.
  2. **Example**: If a record in the Orders table refers to a CustomerID in the Customers table, the foreign key constraint ensures that no order is associated with a non-existent customer, preventing orphan records.

**Domain Integrity**:

* 1. Domain integrity ensures that all values entered into a database column comply with the defined data type, constraints, and permissible range of values.
  2. **Example**: A column for storing dates ensures that only valid date values (e.g., '2024-01-01') are entered, preventing the accidental insertion of invalid data like 'XYZ' into the date column.

**Check Constraints**:

* 1. Check constraints ensure that data adheres to specific conditions, such as a value falling within a particular range or meeting a logical requirement. This prevents accidental data that doesn't meet the business logic or rules.
  2. **Example**: A salary field may have a check constraint to ensure that the salary is greater than 0, preventing accidental entry of negative salaries.

By enforcing these constraints, integrity constraints help maintain the correctness of the data, prevent anomalies, and protect the database from accidental damage due to invalid data entries, deletions, or updates.

1. SQL offers five built in aggregate functions, list them. Explain aggregate with grouping.

### ****SQL Built-in Aggregate Functions (5 Marks)****

SQL provides several **aggregate functions** that operate on a set of values and return a single result. These functions are typically used with the **GROUP BY** clause to group rows that share a common property. The five built-in aggregate functions are:

**COUNT()**:

* 1. Returns the number of rows in a set. It counts the number of non-null values in a column.
  2. **Example**: COUNT(\*) counts all rows, while COUNT(column\_name) counts non-null entries in a specific column.

**SUM()**:

* 1. Adds up the values in a numeric column.
  2. **Example**: SUM(salary) would return the total salary for all employees.

**AVG()**:

* 1. Calculates the average value of a numeric column.
  2. **Example**: AVG(age) would return the average age from the "age" column.

**MAX()**:

* 1. Returns the maximum value in a column.
  2. **Example**: MAX(price) would return the highest price in a "price" column.

**MIN()**:

* 1. Returns the minimum value in a column.
  2. **Example**: MIN(price) would return the lowest price in a "price" column.

### ****Aggregate with Grouping:****

When using aggregate functions with **GROUP BY**, you group rows based on one or more columns and then apply the aggregate function to each group. This allows for performing calculations (like sums or averages) on subsets of data within the table.

**Syntax**:

sql

Copy code

SELECT column\_name, AGGREGATE\_FUNCTION(column\_name)

FROM table\_name

GROUP BY column\_name;

**Example**: Suppose you have a table employees with the following columns: department\_id, employee\_id, and salary. To find the total salary per department, you would use:

SELECT department\_id, SUM(salary) AS total\_salary

FROM employees

GROUP BY department\_id;

In this query:

* + The GROUP BY department\_id groups the rows by each department.
  + The SUM(salary) calculates the total salary within each department.

**Explanation**:

* **GROUP BY** is essential when you want to perform an aggregate calculation on data that is grouped by a certain criterion (e.g., by department, by region, by product).
* The **aggregate functions** (like SUM(), AVG(), COUNT()) will then operate on each group individually, providing insight into each subset of data rather than the whole dataset.

This combination of **aggregation** and **grouping** allows users to summarize and analyze data efficiently, gaining insights from large datasets without needing to process each record individually.

1. **Differentiate between having and where clause in SQL**

### ****Difference Between**** HAVING ****and**** WHERE ****Clause in SQL (3 Marks)****

**Function**:

* 1. WHERE **Clause**: Filters rows before aggregation (used for filtering individual records).
  2. HAVING **Clause**: Filters groups after aggregation (used for filtering aggregated data).

**Usage with Aggregates**:

* 1. WHERE: Cannot be used with aggregate functions (e.g., SUM(), COUNT()), works on individual rows.
  2. HAVING: Used with aggregate functions to filter groups after GROUP BY is applied.

**Example**:

**Where:**

SELECT department, salary

FROM employees

WHERE salary > 5000;

**Having:**

SELECT department, AVG(salary)

FROM employees

GROUP BY department

HAVING AVG(salary) > 5000;

### Summary:

* WHERE filters rows before grouping.
* HAVING filters rows after grouping and aggregation.

**7 No.**

1. Classify and explain the failures occurred in database systems?

### ****Classifications of Failures in Database Systems (4 Marks)****

Failures in a database system can be classified into the following types:

### 1. ****Transaction Failures****:

* **Explanation**: These occur when a transaction cannot complete due to an error, such as logical inconsistencies, invalid data, or violations of integrity constraints. The transaction may be aborted, or a rollback may be required to maintain consistency.
* **Example**: A banking transaction fails when a user tries to withdraw more money than available in their account.

### 2. ****System Failures****:

* **Explanation**: These happen when the DBMS or its underlying hardware (e.g., server or OS) crashes. This type of failure may result in the loss of uncommitted data, requiring recovery procedures to restore the system to a consistent state.
* **Example**: A power outage or a server crash causes the database to stop functioning, potentially losing unsaved transaction data.

### 3. ****Media Failures****:

* **Explanation**: These failures are related to the physical storage devices, such as hard drives or network storage, where data is stored. Media failures lead to data corruption or loss of stored information.
* **Example**: A disk crash causing the loss of data stored on the disk, resulting in corruption of the database files.

### 4. ****Application Failures****:

* **Explanation**: These occur when an external application interacting with the database encounters issues such as incorrect query execution or communication errors. This can lead to inconsistent or incomplete data being stored in the database.
* **Example**: A software application fails to correctly update the database, causing inconsistencies in user records.

### ****Conclusion****:

Different types of failures (transaction, system, media, and application failures) can affect the integrity and availability of data in a database. Handling and recovery mechanisms such as backups, transaction logs, and consistency checks are critical to mitigating the impact of these failure

1. Draw a simple abstract transaction model where all possibilities transaction state must be present and explain each of them.

### ****Abstract Transaction Model****

In a database system, a **transaction** represents a unit of work that must be completed fully or not at all, ensuring the integrity and consistency of the database. A transaction can be in one of several states during its lifecycle. Below is the **abstract transaction model** with all possible transaction states and their explanations.

#### ****Transaction States in DBMS****

+-----------------+

| Active |

+-----------------+

|

v

+-----------------+ +-----------------+

| Partially |------>| Committed |

| Committed | | |

+-----------------+ +-----------------+

|

v

+-----------------+

| Aborted |

+-----------------+

### ****1. Active State****:

* **Explanation**:
  + This is the initial state of the transaction. A transaction is active when it is being executed and has not yet completed. During this state, the transaction may read and modify the database.
  + The transaction can still be committed (if it finishes successfully) or aborted (if it encounters an error).

### ****2. Partially Committed State****:

* **Explanation**:
  + This state occurs after the **commit** operation has been issued but before the changes made by the transaction are fully written to the database.
  + In this state, all operations of the transaction have been completed successfully, but the database has not yet been updated to reflect these changes permanently.

### ****3. Committed State****:

* **Explanation**:
  + A transaction enters this state once it has successfully completed its operations and has been permanently recorded in the database.
  + Once the transaction reaches this state, its changes are permanent, and the transaction is considered **successful**.
  + At this point, the transaction is considered **committed**, and no rollback is possible.

### ****4. Aborted State****:

* **Explanation**:
  + This state occurs when a transaction cannot be completed successfully due to an error or a failure in execution. In this case, the system will roll back any changes made during the transaction, and the transaction will not be able to affect the database.
  + A transaction may enter the **aborted** state if a system failure occurs or if an explicit **rollback** command is issued.

### ****Explanation of the States in the Model****:

* **Active State**: The transaction is in progress and performing its operations (e.g., reading or writing data).
* **Partially Committed State**: The transaction has finished executing, and the changes are ready to be committed to the database but haven't been fully saved yet.
* **Committed State**: The changes made by the transaction are saved permanently, and the transaction is considered completed successfully.
* **Aborted State**: The transaction is terminated due to failure, and any changes made are discarded to maintain database consistency.

### ****State Transitions****:

* A transaction begins in the **Active** state.
* After completing its operations, it moves to **Partially Committed** state, where it awaits confirmation.
* If all conditions are met, the transaction moves to **Committed** state.
* If an error occurs or if the transaction fails at any point, it enters the **Aborted** state, and the system performs a rollback.

This model helps ensure that the database remains in a consistent state by handling all possible transaction states properly.

1. What are the purpose of the word commit and rollback in DBMS?

· **Commit**:

* **Purpose**: The **commit** command is used to make all changes made during the current transaction permanent. Once a transaction is committed, its changes are saved to the database, and the transaction is considered successfully completed.

· **Rollback**:

* **Purpose**: The **rollback** command is used to undo all changes made during the current transaction. It restores the database to the state it was in before the transaction started, typically used in the case of errors or failures.

**8 No.**

1. What do you understand by ACID properties of transaction? Explain properties with proper example?

### ****ACID Properties of Transactions****

ACID stands for **Atomicity**, **Consistency**, **Isolation**, and **Durability**, which are a set of properties that guarantee reliable processing of database transactions. These properties ensure that the database remains in a consistent state, even in the event of system failures.

### ****1. Atomicity****:

**Definition**: Atomicity ensures that a transaction is treated as a single, indivisible unit. Either all operations within the transaction are executed, or none of them are. If a failure occurs, the transaction will be rolled back, and no partial changes will be saved to the database.

**Example**: Consider a bank transaction where $100 is transferred from Account A to Account B. Atomicity ensures that either both the debit from Account A and the credit to Account B happen, or neither happens (i.e., the transaction is rolled back in case of an error, ensuring no money is lost or incorrectly transferred).

### ****2. Consistency****:

**Definition**: Consistency ensures that a transaction takes the database from one valid state to another valid state. The integrity constraints, such as foreign keys or unique keys, must be maintained throughout the transaction. If a transaction violates these constraints, it will be rolled back, maintaining the database's consistency.

**Example**: If a bank has a rule that no account can go below a zero balance, consistency ensures that after the transaction, both accounts involved in the transaction must still satisfy this rule. For example, if Account A has a balance of $50, the transaction cannot transfer $100 out of it.

### ****3. Isolation****:

**Definition**: Isolation ensures that the operations of one transaction are isolated from those of other concurrent transactions. Even if multiple transactions are executed simultaneously, each transaction will execute as if it were the only one running, preventing interference with each other. The results of intermediate steps in a transaction are not visible to other transactions until the transaction is committed.

**Example**: If two transactions are running simultaneously, one transferring money from Account A to Account B and the other from Account B to Account C, isolation ensures that no other transaction can see the intermediate results (e.g., Account B’s balance being temporarily incorrect) until both transactions are committed.

### ****4. Durability****:

**Definition**: Durability ensures that once a transaction has been committed, its changes are permanent, even in the event of a system failure, such as a crash. The database is guaranteed to reflect the committed transaction after a restart or recovery.

**Example**: After successfully transferring $100 from Account A to Account B, durability ensures that the changes are permanently recorded in the database, and even if the system crashes right after the commit, the transaction's effect (the updated balances) will be retained upon recovery.

### ****Summary of ACID Properties with Example****:

1. **Atomicity**: All or nothing. If a bank transfer fails halfway, no money is lost.
2. **Consistency**: The database adheres to all rules. If the system allows negative balances, a transfer cannot push an account below zero.
3. **Isolation**: Transactions don't interfere with each other. One user's transaction won't affect another user's transaction in the interim.
4. **Durability**: Once a transaction is committed, it cannot be undone, even if the system crashes afterward. The data changes will persist.

These properties together ensure that database transactions are processed reliably and maintain the integrity of the database.

1. Multiple transaction are allowed to run concurrently in a system, what are the advantages and challenges to do that?

### ****Advantages of Concurrent Transactions****:

**Improved System Utilization**:

* 1. **Advantage**: Allowing multiple transactions to run concurrently helps maximize the utilization of system resources, such as CPU and I/O, leading to better performance and faster processing times. This is especially useful in multi-core processors, where parallel processing can occur.

**Increased Throughput**:

* 1. **Advantage**: By executing multiple transactions simultaneously, the database system can handle a larger volume of transactions in a shorter period of time, improving overall throughput and system efficiency.

**Faster Response Times**:

* 1. **Advantage**: Concurrent transactions can lead to faster response times for users, as transactions are processed in parallel. For example, while one transaction is waiting for disk I/O, another can be processed, thus reducing waiting times.

**Better Resource Utilization**:

* 1. **Advantage**: By allowing concurrent execution, idle resources (such as CPU cycles) can be used more effectively, preventing system resources from being wasted.

### ****Challenges of Concurrent Transactions****:

**Data Integrity Issues**:

* 1. **Challenge**: When multiple transactions access and modify the same data concurrently, there is a risk of data inconsistencies, such as **lost updates**, **temporary inconsistency**, or **uncommitted data** being read. Without proper mechanisms, such as locking or isolation levels, the integrity of the data can be compromised.

**Deadlock**:

* 1. **Challenge**: Concurrent transactions can lead to **deadlocks**, where two or more transactions are waiting for each other to release locks on resources. This can cause a situation where none of the transactions can proceed, leading to a system halt.

**Complexity in Transaction Management**:

* 1. **Challenge**: Managing multiple concurrent transactions increases the complexity of the database management system (DBMS). Ensuring correct synchronization, applying isolation levels, and resolving conflicts requires additional processing overhead and sophisticated transaction management techniques, such as locking mechanisms or timestamps.

**Resource Contention**:

* 1. **Challenge**: Multiple transactions may compete for the same resources (e.g., memory, CPU, disk), leading to **resource contention**. This can result in delays or slow performance if not handled properly, affecting the overall system efficiency.

### ****Conclusion****:

While allowing concurrent transactions in a database system provides several advantages in terms of system performance, throughput, and resource utilization, it also introduces challenges related to data integrity, deadlock, and complexity in transaction management. Proper concurrency control mechanisms, such as locking, isolation levels, and transaction management, are essential to address these challenges.

1. Define conflict equivalent and conflict serializable.

### ****Conflict Equivalent****:

* **Definition**: Two schedules (or transaction executions) are **conflict equivalent** if they consist of the same operations, and the operations that conflict (i.e., read-write, write-read, or write-write on the same data item) appear in the same order in both schedules. In other words, the relative order of conflicting operations must be preserved.

### ****Conflict Serializable****:

* **Definition**: A schedule is **conflict serializable** if it is conflict equivalent to some serial schedule (a schedule in which transactions are executed one after the other without overlapping). Conflict serializability ensures that, despite transactions being interleaved, the final result will be the same as if the transactions were executed serially, one after the other.

**14-Batch**

**1 no.**

1. Define the terms instance and schema

**Instance:**An instance refers to the actual data stored in a database at a specific moment in time. It represents a snapshot of the database's state, including all records and their current values.

**Schema:**A schema is the overall logical structure of a database, defining the tables, fields, relationships, constraints, and data types. It provides the blueprint for how data is organized and accessed.

1. Write the major advantages and disadvantages of the database system

**Advantages of a Database System:**

1. **Data Consistency and Integrity:** Centralized management ensures uniformity and accuracy across the database.
2. **Data Security:** Access controls and authentication prevent unauthorized access.
3. **Data Sharing:** Multiple users can access data simultaneously without conflicts.
4. **Reduced Data Redundancy:** Centralized storage avoids duplicate data, saving storage space.
5. **Improved Data Access:** Efficient query and retrieval mechanisms enable quick access to data.
6. **Backup and Recovery:** Data can be restored after a failure, ensuring minimal data loss.
7. **Concurrency Control:** Multiple users can access and modify data concurrently without conflicts.

**Disadvantages of a Database System:**

1. **Complexity:** Design, implementation, and maintenance require specialized knowledge.
2. **Cost:** High expenses related to software, hardware, and skilled personnel.
3. **Performance Overhead:** Complex operations may slow down access for small applications.
4. **Risk of Data Loss:** Centralized storage increases vulnerability if the system fails without backups.
5. **Maintenance:** Regular updates, backups, and system checks require ongoing effort and resources.
6. **Data Migration Issues:** Transferring data between systems can be complex and time-consuming.
7. **Potential Downtime:** Maintenance activities or unexpected failures can disrupt data access and availability.
8. What are the responsibilities of database management systems?

**Responsibilities of Database Management Systems (DBMS):**

1. **Data Storage and Retrieval:** Efficiently store, organize, and retrieve data.
2. **Data Security:** Ensure data protection through access control and authentication.
3. **Data Integrity:** Maintain data accuracy and consistency using constraints.
4. **Concurrency Control:** Manage simultaneous access by multiple users.
5. **Backup and Recovery:** Provide mechanisms to recover data in case of failures.
6. Explain the different types of database system users?

**Types of Database System Users:**

**Database Administrators (DBAs):**  
DBAs are responsible for the overall management of the database system. They handle tasks such as database design, user access control, performance tuning, backups, and recovery. They ensure that the database is secure, efficient, and reliable.

**End Users:**  
End users are individuals who interact with the database to perform specific tasks such as querying, data entry, or report generation. They may use applications with a graphical user interface (GUI) to access the database, but they do not manage or maintain the system.

**Application Programmers:**  
Application programmers design and develop applications that interact with the database. They write programs to insert, update, and retrieve data from the database, typically using languages like SQL or other database management languages.

**System Analysts:**  
System analysts focus on analyzing and designing the requirements of the database system. They act as intermediaries between the end users and the DBAs, helping to define how data should be structured and accessed.

**Data Analysts:**  
Data analysts are responsible for analyzing data stored in the database to extract meaningful insights. They often use data mining techniques and business intelligence tools to generate reports and analyze trends.

2 no.

1. Write down the difference between DDL and DML?

**Difference between DDL (Data Definition Language) and DML (Data Manipulation Language):**

| **Aspect** | **DDL (Data Definition Language)** | **DML (Data Manipulation Language)** |
| --- | --- | --- |
| **Purpose** | Used to define and modify the database structure. | Used to manipulate data within the database (insert, update, delete). |
| **Commands** | CREATE, ALTER, DROP, TRUNCATE, RENAME. | SELECT, INSERT, UPDATE, DELETE. |
| **Effect on Data** | Does not affect the data; it only affects the database schema. | Affects the actual data stored in the database. |
| **Transaction Control** | No transaction control; changes are permanent and immediate. | Supports transaction control (commits, rollbacks). |
| **Execution Frequency** | Executed less frequently (usually during database design). | Executed frequently by users or applications during runtime. |
| **Usage** | Primarily used by database administrators for schema management. | Primarily used by end users and applications for data manipulation. |
| **Impact on Performance** | Can impact the overall structure and performance of the database. | Affects the performance based on the amount of data being manipulated. |
| **Permissions** | Requires higher-level permissions (DBA access) to execute. | Typically requires lower-level permissions (user-level access). |

1. List four significant differences between a file processing system and a DBMS?

**Differences between a File Processing System and a DBMS:**

| **Aspect** | **File Processing System** | **DBMS (Database Management System)** |
| --- | --- | --- |
| **Data Storage** | Data is stored in separate files with minimal organization. | Data is stored centrally in a structured format (tables). |
| **Data Redundancy** | High data redundancy due to multiple copies of data in different files. | Minimal data redundancy with normalization and centralized storage. |
| **Data Access** | Accessing data requires custom programs for each file. | Provides a unified query interface (e.g., SQL) for easy data retrieval. |
| **Data Integrity** | Ensures data integrity manually through application logic. | DBMS enforces data integrity through constraints and rules (e.g., primary keys, foreign keys). |

1. Define different state of a transaction?

**Different States of a Transaction:**

**New (or Active):**  
In this state, the transaction has been initiated but has not yet started execution. The system has allocated resources for the transaction, and it is ready to perform operations.

**Running (or Active):**  
The transaction is actively being processed. During this state, the database operations (such as insert, update, or delete) are being executed. The transaction can be in progress and performing work.

**Committed:**  
Once all operations of the transaction are successfully completed, the transaction enters the committed state. The changes made to the database are permanently saved and cannot be rolled back.

**Aborted (or Failed):**  
If an error occurs during the transaction execution, or if the transaction is intentionally rolled back due to an issue (such as violation of constraints), the transaction enters the aborted state. In this state, all changes made by the transaction are undone, and the database is returned to its previous consistent state.

These states ensure that transactions follow the **ACID properties** (Atomicity, Consistency, Isolation, Durability) to maintain the integrity of the database.

**3 no.**

1. Describe the responsibilities of a storage manager?

**Responsibilities of a Storage Manager:**

**Data Storage Management:**  
The storage manager is responsible for managing how data is stored on physical storage devices (like hard disks or SSDs). It ensures that data is stored efficiently, considering factors like access speed and storage space utilization.

**Data Retrieval and Access:**  
It handles the reading and writing of data to storage, ensuring that data can be quickly retrieved when needed by the DBMS. This includes managing buffer pools and caching strategies for faster access.

**File Organization and Indexing:**  
The storage manager organizes data into files and maintains indexes to speed up query processing. It decides how records are arranged within the files and which indexing techniques (e.g., B-trees) are used to ensure efficient data retrieval.

**Backup and Recovery:**  
The storage manager is responsible for ensuring that regular backups of the database are made and that recovery mechanisms are in place. If a failure occurs, it ensures the ability to restore the database to its last consistent state.

These responsibilities ensure that the data is stored securely, accessed efficiently, and recoverable in case of failure.

1. Consider the following expressions, which use the result of relational algebra operation as input to another operation. For each expression, explain in words what the expression does.

(i) σ year >= 2009(takes) ⋈ student

(ii) σ year>=2009(takes ⋈ student)

(iii) π ID, name, course\_id (student ⋈ takes)

**) σ year >= 2009(takes) ⋈ student**

**σ year >= 2009(takes):**  
This is a **selection** operation (σ) applied to the "takes" relation. It filters the rows where the value of the year attribute is greater than or equal to 2009. So, it selects only the records from the "takes" table where students have enrolled in courses in 2009 or later.

**⋈ student:**  
After filtering the "takes" table, the result is then **joined** (⋈) with the "student" table. The default assumption is that the join is based on a common attribute, typically ID (student ID), which is present in both the "takes" and "student" tables.

**What it does:**  
This expression first filters the "takes" relation to include only the rows where year >= 2009, and then it joins this filtered set with the "student" table based on the student ID. The result is a relation that includes information about students who have taken courses in 2009 or later, with details from both the "takes" and "student" tables.

**(ii) σ year >= 2009(takes ⋈ student)**

**takes ⋈ student:**  
This is a **join** operation between the "takes" and "student" relations, based on a common attribute (likely ID). The result is a relation that combines data from both the "takes" and "student" tables.

**σ year >= 2009:**  
After performing the join, the **selection** operation (σ) is applied to the result. This filters the rows where the year attribute from the "takes" table is greater than or equal to 2009.

**What it does:**  
This expression first joins the "takes" and "student" tables, combining student information with their corresponding course enrollments. Then, it filters this combined result to only include rows where the course year is 2009 or later. The result is a relation showing the details of students enrolled in courses from 2009 onward.

**(iii) π ID, name, course\_id (student ⋈ takes)**

**student ⋈ takes:**  
This is a **join** operation between the "student" and "takes" tables based on a common attribute (likely ID). The result is a relation containing data from both tables, where each record links a student to their corresponding course enrollment.

**π ID, name, course\_id:**  
This is a **projection** operation (π) that selects specific columns from the result of the join. Here, it selects only the ID, name, and course\_id attributes, ignoring other attributes that may be present in the result of the join.

**What it does:**  
This expression joins the "student" and "takes" tables and then projects (selects) only the ID, name (of the student), and course\_id (of the course). The result is a relation that shows the student ID, student name, and the course IDs they are enrolled in.

1. Define aggregate function?

**Aggregate Function** refers to a type of function that performs a calculation on a set of values and returns a single value. It is commonly used in SQL and relational databases to summarize or analyze data.

Common examples of aggregate functions include:

1. **COUNT:** Counts the number of rows or values in a column.
2. **SUM:** Adds up the values in a column.
3. **AVG:** Calculates the average of values in a column.
4. **MIN:** Returns the smallest value in a column.
5. **MAX:** Returns the largest value in a column.

**4 No.**

1. Explain the distinction among the terms primary key, foreign key, candidate key and super key?

### 1. ****Primary Key:****

* **Definition:** A primary key is a column or a set of columns in a table that uniquely identifies each row in that table. It cannot have NULL values, and its values must be unique across all rows.
* **Purpose:** Ensures data integrity by uniquely identifying records in a table.
* **Example:** In a **Student** table, the student\_id column can be the primary key as it uniquely identifies each student.

### 2. ****Foreign Key:****

* **Definition:** A foreign key is a column or a set of columns in a table that creates a link between two tables. It refers to the primary key in another table and establishes a relationship between the two tables.
* **Purpose:** Ensures referential integrity between two tables. It maintains consistency by enforcing that the value in the foreign key column must exist in the referenced primary key column.
* **Example:** In an **Enrollment** table, a student\_id column might be a foreign key that references the student\_id primary key in the **Student** table.

### 3. ****Candidate Key:****

* **Definition:** A candidate key is any column or set of columns that could be used as a primary key. Each candidate key must be able to uniquely identify rows in a table and should not contain NULL values. There can be multiple candidate keys in a table.
* **Purpose:** Provides multiple options for uniquely identifying records, from which one can be chosen as the primary key.
* **Example:** In a **Student** table, both student\_id and email could be candidate keys since both uniquely identify a student.

### 4. ****Super Key:****

* **Definition:** A super key is a set of one or more columns that can uniquely identify rows in a table. A super key may contain additional attributes beyond what is necessary for uniqueness. All primary keys and candidate keys are super keys, but not all super keys are candidate keys.
* **Purpose:** Represents a set of columns that can uniquely identify records, but it may contain extra attributes that aren't necessary.
* **Example:** In a **Student** table, the combination of student\_id and email could be a super key, though only student\_id alone would be a candidate key.

### ****Summary of Differences:****

| **Key** | **Uniqueness** | **Null Values** | **Purpose** | **Example** |
| --- | --- | --- | --- | --- |
| **Primary Key** | Unique | No NULL values | Uniquely identifies each row in the table. | student\_id in the **Student** table. |
| **Foreign Key** | Not necessarily unique | Can be NULL if not applicable | Links one table to another, ensuring referential integrity. | student\_id in **Enrollment** referencing **Student** table. |
| **Candidate Key** | Unique | No NULL values | Any column(s) that could serve as the primary key. | student\_id or email in the **Student** table. |
| **Super Key** | Unique | No NULL values | Any set of columns that can uniquely identify a row, possibly with extra attributes. | student\_id + email in the **Student** table. |

**5 NO.**

1. Consider the bank database. Give an expression in the relational algebra for each of the following queries:  
     
   branch(branch\_name, branch city, assets)  
   customer(customer\_name, customer street, customer city)  
   loan(loan\_number, branch name, amount)  
   borrower(customer\_name, loan number)  
   account(account number, branch name, balance)  
   depositor(customer\_name, account number)  
     
   (i) Given your choice of primary keys, identify appropriate foreign keys.  
   (ii) find the names of the branches located in “Chicago”  
   (iii) Find the names of all borrowers who have a loan in branch “Downtown”.  
   (iv) Find the names of all depositors who have an account with a value greater than $6,000 at the “Uptown” branch.  
   (v) find all loan numbers with a loan value greater than $10,000.

### (i) ****Primary Keys and Foreign Keys Identification****

**Primary Keys:**

* branch(branch\_name) (assuming branch\_name uniquely identifies each branch)
* customer(customer\_name) (assuming customer\_name uniquely identifies each customer)
* loan(loan\_number) (assuming loan\_number uniquely identifies each loan)
* account(account\_number) (assuming account\_number uniquely identifies each account)

**Foreign Keys:**

* loan(branch\_name) is a **foreign key** referencing branch(branch\_name) to represent the relationship between loans and branches.
* borrower(customer\_name, loan\_number) has:
  + customer\_name as a **foreign key** referencing customer(customer\_name) to represent the relationship between customers and loans.
  + loan\_number as a **foreign key** referencing loan(loan\_number) to link loans and borrowers.
* account(branch\_name) is a **foreign key** referencing branch(branch\_name) to represent the relationship between accounts and branches.
* depositor(customer\_name, account\_number) has:
  + customer\_name as a **foreign key** referencing customer(customer\_name) to link customers and accounts.
  + account\_number as a **foreign key** referencing account(account\_number) to represent the relationship between accounts and depositors.

### (ii) ****Find the names of the branches located in “Chicago”****

To find the branch names located in "Chicago," we can use the **selection (σ)** operation, followed by **projection (π)** to extract the branch\_name column.

**Relational Algebra Expression:**

π branch\_name (σ branch\_city = 'Chicago' (branch))

* **Explanation:**
  + **σ branch\_city = 'Chicago' (branch)** selects rows from the branch table where the branch\_city is "Chicago."
  + **π branch\_name** projects only the branch\_name column from the selected rows.

### (iii) ****Find the names of all borrowers who have a loan in branch “Downtown”****

To find borrowers with loans in the "Downtown" branch, we need to join the borrower, loan, and branch tables, and apply the condition for the "Downtown" branch.

**Relational Algebra Expression:**

π customer\_name (σ branch\_name = 'Downtown' (borrower ⋈ loan))

* **Explanation:**
  + **borrower ⋈ loan** joins the borrower and loan tables on loan\_number.
  + **σ branch\_name = 'Downtown'** applies the selection operation to filter rows where the branch\_name is "Downtown."
  + **π customer\_name** projects the customer\_name column, which gives the names of borrowers.

### (iv) ****Find the names of all depositors who have an account with a balance greater than $6,000 at the “Uptown” branch****

To find the names of depositors with accounts over $6,000 at the "Uptown" branch, we need to join the depositor, account, and branch tables, apply the conditions for balance and branch name, and then project the customer names.

**Relational Algebra Expression:**

π customer\_name (σ balance > 6000 AND branch\_name = 'Uptown' (depositor ⋈ account))

* **Explanation:**
  + **depositor ⋈ account** joins the depositor and account tables on account\_number.
  + **σ balance > 6000 AND branch\_name = 'Uptown'** applies the selection condition to find depositors with an account balance greater than $6,000 in the "Uptown" branch.
  + **π customer\_name** projects the customer\_name column to get the names of the depositors.

### (v) ****Find all loan numbers with a loan value greater than $10,000****

To find loan numbers with a loan value greater than $10,000, we can apply the **selection (σ)** operation on the loan table, followed by **projection (π)** to extract the loan\_number.

**Relational Algebra Expression:**

π loan\_number (σ amount > 10000 (loan))

* **Explanation:**
  + **σ amount > 10000 (loan)** selects rows from the loan table where the amount is greater than $10,000.
  + **π loan\_number** projects the loan\_number column to get the list of loan numbers that meet the condition.

### Summary of Expressions:

**Primary Keys & Foreign Keys:**

* + Primary keys are branch\_name, customer\_name, loan\_number, and account\_number.
  + Foreign keys include references between tables such as loan(branch\_name), borrower(customer\_name, loan\_number), account(branch\_name), and depositor(customer\_name, account\_number).

**Branches in Chicago:**

π branch\_name (σ branch\_city = 'Chicago' (branch))

**Borrowers in the Downtown branch:**

π customer\_name (σ branch\_name = 'Downtown' (borrower ⋈ loan))

**Depositors with accounts > $6,000 at Uptown branch:**

π customer\_name (σ balance > 6000 AND branch\_name = 'Uptown' (depositor ⋈ account))

**Loan numbers with loan value > $10,000:**

π loan\_number (σ amount > 10000 (loan))

**6 no.**

1. Why “with”clause is needed for any database?

The **WITH** clause, also known as **Common Table Expressions (CTE)**, is essential in SQL for several reasons, especially when dealing with complex queries. Here are the key reasons why the **WITH** clause is needed in any database:

### 1. ****Improved Readability:****

* The **WITH** clause allows the definition of temporary result sets (CTEs) that can be referenced within the main query. This makes queries more readable by breaking them into smaller, more understandable parts.
* Without it, queries become difficult to read, especially when subqueries are used repeatedly.

### 2. ****Avoiding Repetition of Subqueries:****

* CTEs defined using the **WITH** clause allow you to write a subquery once and reference it multiple times within the same query. This reduces redundancy and the chances of errors, making the query more efficient.

### 3. ****Simplifying Complex Queries:****

* The **WITH** clause helps in simplifying complex queries by separating the logic into different sections. Each CTE can be thought of as a building block that contributes to the overall result, improving query organization and making debugging easier.

### 4. ****Improved Query Performance:****

* In some cases, using the **WITH** clause can enhance performance by preventing the database from recalculating the same result set multiple times, especially in cases where the same subquery needs to be reused several times within the query.

1. Write down a SQL query to create and drop a table in a database?

#### ****SQL Query to Create a Table:****

To create a table in a database, the CREATE TABLE statement is used. Below is an example SQL query to create a table:

**Example:**

CREATE TABLE Students (

student\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

date\_of\_birth DATE,

enrollment\_date DATE

);

**Explanation:**

* CREATE TABLE Students: This part creates a new table named "Students".
* student\_id INT PRIMARY KEY: This defines the student\_id column, which is of type INT and will be the primary key of the table.
* first\_name VARCHAR(50): This defines the first\_name column with a maximum length of 50 characters.
* last\_name VARCHAR(50): This defines the last\_name column with a maximum length of 50 characters.
* date\_of\_birth DATE: This defines the date\_of\_birth column to store date values.
* enrollment\_date DATE: This defines the enrollment\_date column to store date values.

#### 2. ****SQL Query to Drop a Table:****

To delete a table from the database, the DROP TABLE statement is used.

**Example:**

DROP TABLE Students;

**Explanation:**

* DROP TABLE Students: This command deletes the "Students" table from the database, along with all its data and structure.

1. Write down the responsibilities of a DBA?

### Responsibilities of a Database Administrator (DBA) (4 Marks)

A **Database Administrator (DBA)** is responsible for managing and maintaining databases to ensure they function efficiently and securely. Here are the key responsibilities:

**Database Design and Implementation:**

* 1. The DBA is responsible for designing the database schema, tables, and relationships according to the requirements of the organization.
  2. They also ensure that the database is structured in a way that allows efficient storage and retrieval of data.

**Database Security:**

* 1. The DBA ensures that the database is secure by managing user access, permissions, and roles. They implement encryption and ensure data privacy.
  2. They also monitor for unauthorized access or breaches and apply necessary security patches.

**Backup and Recovery:**

* 1. The DBA designs and implements backup and recovery strategies to protect data from loss or corruption.
  2. They ensure regular backups are taken and can restore the database to its previous state in case of failures or disasters.

**Performance Tuning and Optimization:**

* 1. The DBA is responsible for monitoring database performance, analyzing queries, and optimizing them for better efficiency.
  2. They tune the database server settings, indexes, and queries to improve speed and resource utilization.

In summary, a **DBA** is responsible for designing, securing, maintaining, and optimizing databases to ensure data availability, integrity, and performance.

**7 No.  
  
(A)** What are the basic domain types in SQL?

### Basic Domain Types in SQL (2 Marks)

In SQL, **domain types** refer to the set of allowable values that a column can hold. The basic domain types are as follows:

**Numeric Types:**

* 1. These types are used to store numbers. Common numeric types include:
     1. INT: Integer values.
     2. DECIMAL(p, s): Fixed-point numbers, where p is the precision and s is the scale.
     3. FLOAT, DOUBLE: Floating-point numbers with varying precision.

**Character and String Types:**

* 1. These types are used to store alphanumeric data (text):
     1. CHAR(n): Fixed-length character string with n characters.
     2. VARCHAR(n): Variable-length character string with a maximum length of n characters.
     3. TEXT: Variable-length text.

**Date and Time Types:**

* 1. These types are used to store dates and time-related data:
     1. DATE: Stores date values (year, month, day).
     2. TIME: Stores time values (hour, minute, second).
     3. DATETIME: Stores both date and time.

These domain types define the type of data that can be stored in each column of a database table.

1. What is transaction? Describe the ACID properties of the transaction?

### What is a Transaction? (1 Mark)

A **transaction** in a database is a sequence of one or more operations (such as read, write, update, or delete) that are executed as a single unit. A transaction must be processed entirely or not at all. It ensures data integrity and consistency even in cases of system failures or errors.

### ACID Properties of a Transaction (4 Marks)

The **ACID** properties ensure that database transactions are processed reliably and guarantee the consistency of the database. **ACID** stands for:

**Atomicity:**

* 1. **Definition:** Atomicity ensures that a transaction is treated as a single unit of work, meaning it is either fully completed or not executed at all.
  2. **Explanation:** If any part of the transaction fails, the entire transaction is rolled back, and the database remains unchanged, maintaining its integrity.
  3. **Example:** If a bank transfer involves transferring money from one account to another, atomicity ensures that either both accounts are updated, or neither is updated if any error occurs.

**Consistency:**

* 1. **Definition:** Consistency ensures that a transaction takes the database from one valid state to another valid state, preserving the integrity of the database.
  2. **Explanation:** The database must always meet predefined rules, such as constraints, triggers, and referential integrity, before and after the transaction.
  3. **Example:** If a database has a rule that the balance of an account cannot be negative, consistency ensures that this rule is not violated during the transaction.

**Isolation:**

* 1. **Definition:** Isolation ensures that transactions are executed in isolation from one another, meaning that the intermediate state of a transaction is not visible to other transactions.
  2. **Explanation:** This prevents transactions from interfering with each other and ensures that the final outcome of each transaction is independent of the execution order.
  3. **Example:** If two users are transferring money at the same time, isolation ensures that they don’t both access the same account at the same time, potentially causing inconsistent data.

**Durability:**

* 1. **Definition:** Durability ensures that once a transaction is committed, its changes are permanent and will not be lost, even in the event of a system failure.
  2. **Explanation:** After the transaction completes, the changes made to the database are stored permanently, ensuring data persistence.
  3. **Example:** After completing a bank transfer, even if there is a system crash, the updated account balances will still be reflected once the system restarts.

### Summary:

The **ACID** properties (Atomicity, Consistency, Isolation, Durability) ensure that database transactions are reliable, ensuring data integrity and consistency while protecting against errors and system failures.

1. **Define the terms atomicity and durability. Explain the shadow-copy technique for the implementing atomicity and durability?**

### Atomicity (1 Mark)

**Atomicity** refers to the property of a transaction in a database that ensures it is treated as a single, indivisible unit. A transaction is either fully executed (committed) or not executed at all (rolled back). If any part of the transaction fails, the entire transaction is aborted, and the database is left unchanged, ensuring no partial updates are made.

**Example:** In a banking transaction transferring money from one account to another, atomicity ensures that both the withdrawal and deposit operations either complete successfully or neither occurs.

### Durability (1 Mark)

**Durability** ensures that once a transaction is committed, its changes are permanent, even if the system crashes immediately afterward. The data modifications made during the transaction are saved in non-volatile storage, so the changes remain in the database permanently, even in case of system failures.

**Example:** If a transaction that updates an account balance is successfully committed, even if there is a power failure afterward, the updated balance will still be present in the database upon recovery.

### Shadow-Copy Technique for Implementing Atomicity and Durability (3 Marks)

The **shadow-copy technique** is a method used to implement the **Atomicity** and **Durability** properties of transactions in database systems. It involves creating a **shadow copy** or backup of the data before making any changes during a transaction. Here’s how it works:

**Atomicity via Shadow-Copy:**

* 1. Before any updates are made to the database, the system creates a **shadow copy** (a backup or temporary copy) of the data being modified.
  2. The transaction operates on this shadow copy. If the transaction is successful, the system writes the changes to the original database, making the modifications permanent.
  3. If any error or failure occurs during the transaction, the system can discard the shadow copy, leaving the original data unchanged, thus ensuring **atomicity** (either complete success or no effect at all).

**Durability via Shadow-Copy:**

* 1. Once the transaction is committed, the changes made to the shadow copy are written to the actual database storage (disk).
  2. Even if a failure occurs after committing the transaction, the changes can be recovered from the shadow copy, ensuring that the transaction's effects are not lost.
  3. Since the original data is never directly altered during the transaction, it provides **durability** by ensuring that the committed changes are safely stored.

**Example:**

* Consider a scenario where a bank transaction is updating the balance of two accounts. The database first creates shadow copies of the relevant accounts. If the transaction is successful, the changes are committed to the main database. If the system crashes before committing, the shadow copies are discarded, ensuring no partial changes. If the transaction completes and the system crashes afterward, the committed changes will persist, ensuring durability.

### Summary:

* **Atomicity** ensures a transaction is fully executed or not at all.
* **Durability** guarantees that once a transaction is committed, its changes are permanent.
* The **shadow-copy technique** helps implement both by using temporary copies of data during the transaction and ensuring changes are only written to the original data when the transaction is successful. In case of failure, the shadow copy ensures that the database remains consistent.

**8 No.  
  
(A) Give the difference between Super key and Candidate key?**

### Difference Between Super Key and Candidate Key (2 Marks)

**Super Key:**

* 1. A super key is any set of one or more attributes (columns) that can uniquely identify a row in a table.
  2. It may contain extra attributes that are not required for uniqueness.

**Candidate Key:**

* 1. A candidate key is a minimal super key, meaning it is a super key with no unnecessary attributes.
  2. It uniquely identifies a row and cannot have any redundant attributes.

### Key Difference:

* **Super Key** may have extra attributes, whereas a **Candidate Key** is the minimal set of attributes necessary to uniquely identify a row.

1. **What is integrity constraint? Explain different types of constraints on a single relation?**

### Integrity Constraint (1 Mark)

An **integrity constraint** is a rule that ensures the accuracy and consistency of data within a relational database. It defines the conditions that data in the database must satisfy in order to maintain the integrity and correctness of the database.

### Different Types of Constraints on a Single Relation (3 Marks)

There are several types of constraints that can be applied to a single relation (table) to enforce integrity rules:

**Domain Constraints:**

* 1. **Definition:** Domain constraints specify that the values in a particular column must come from a defined domain or set of allowed values.
  2. **Example:** If the age column is defined as an integer, it ensures that all values in the age column are integers.

**Entity Integrity Constraint:**

* 1. **Definition:** This constraint ensures that every table has a **primary key** and that the values in the primary key are **unique** and **not null**.
  2. **Example:** In a Students table, the student\_id column can be the primary key, ensuring that no two students can have the same ID and that the ID cannot be null.

**Referential Integrity Constraint:**

* 1. **Definition:** This constraint ensures that the foreign key in a table correctly references a primary key in another table. It maintains relationships between tables.
  2. **Example:** In a Courses table, the instructor\_id column might be a foreign key that refers to the instructor\_id in an Instructors table. It ensures that every instructor referenced in the Courses table exists in the Instructors table.

**Check Constraints:**

* 1. **Definition:** Check constraints specify a condition that must be true for every row in the table. It is used to enforce specific business rules.
  2. **Example:** In an Employees table, a check constraint can be used to ensure that the salary column contains values greater than 0 (CHECK (salary > 0)).

**Unique Constraints:**

* 1. **Definition:** Unique constraints ensure that the values in a column (or a set of columns) are unique across all rows in the table. Unlike primary keys, a column with a unique constraint can accept null values.
  2. **Example:** A email column in a Users table might have a unique constraint to ensure no two users have the same email address.

### Summary:

* **Domain constraints** limit the type of data allowed in a column.
* **Entity integrity** ensures each row has a unique, non-null primary key.
* **Referential integrity** enforces correct relationships between tables using foreign keys.
* **Check constraints** define conditions for column values.
* **Unique constraints** ensure that column values are distinct.

1. Consider the bank database. Give an expression in the relational algebra for each of the following queries:  
     
   branch(branch\_name, branch city, assets)  
   customer(customer\_name, customer street, customer city)  
   loan(loan\_number, branch name, amount)  
   borrower(customer\_name, loan number)  
   account(account number, branch name, balance)  
   depositor(customer\_name, account number)  
     
   Find the names of all customers in alphabetic order who have a loan at the Perryridge branch.  
   (i) Find the average account balance at each branch.  
   (ii)Find the branch that has the highest average balance.  
   (iii) Find all the customers who do have both a loan and an account at the bank.

### Relational Algebra Expressions for Bank Database Queries

Given the relations in the bank database:

1. **branch(branch\_name, branch\_city, assets)**
2. **customer(customer\_name, customer\_street, customer\_city)**
3. **loan(loan\_number, branch\_name, amount)**
4. **borrower(customer\_name, loan\_number)**
5. **account(account\_number, branch\_name, balance)**
6. **depositor(customer\_name, account\_number)**

### Queries and Relational Algebra Expressions:

#### (i) Find the names of all customers in alphabetic order who have a loan at the "Perryridge" branch.

**Relational Algebra Expression:**

arduino

Copy code

π customer\_name (σ branch\_name = 'Perryridge' (borrower ⨝ loan))

* **Explanation:**
  + First, perform a selection (σ branch\_name = 'Perryridge') on the loan relation to filter loans related to the "Perryridge" branch.
  + Then, use a natural join (⨝) with the borrower relation to get the corresponding customer names.
  + Finally, project (π customer\_name) the customer names.

#### (ii) Find the average account balance at each branch.

**Relational Algebra Expression:**

scss

Copy code

γ branch\_name; AVG(balance) (account)

* **Explanation:**
  + The **grouping** operation (γ) is applied on the branch\_name attribute, and then the **aggregate function** AVG(balance) is used to calculate the average balance for each branch.

#### (iii) Find the branch that has the highest average balance.

**Relational Algebra Expression:**

scss

Copy code

π branch\_name (σ AVG(balance) = MAX(AVG(balance)) (γ branch\_name; AVG(balance) (account)))

* **Explanation:**
  + First, calculate the average balance for each branch using the same approach as in (ii) with the grouping and aggregation (γ branch\_name; AVG(balance)).
  + Then, apply a selection (σ AVG(balance) = MAX(AVG(balance))) to filter the branch with the highest average balance.
  + Finally, project (π branch\_name) to retrieve the branch name.

#### (iv) Find all the customers who have both a loan and an account at the bank.

**Relational Algebra Expression:**

scss

Copy code

π customer\_name (borrower ⨝ loan) ∩ π customer\_name (depositor ⨝ account)

* **Explanation:**
  + First, join the borrower and loan relations to get all customers with loans (borrower ⨝ loan), and project the customer\_name.
  + Next, join the depositor and account relations to get all customers with accounts (depositor ⨝ account), and project the customer\_name.
  + The **intersection** (∩) operation is used to find customers who appear in both sets (i.e., those who have both a loan and an account).

### Summary of Relational Algebra Operations:

* **Projection (π):** Selects specific columns from a relation.
* **Selection (σ):** Filters rows based on a condition.
* **Join (⨝):** Combines two relations based on matching attribute values.
* **Grouping and Aggregation (γ):** Groups data by specified attributes and applies aggregate functions like AVG.
* **Intersection (∩):** Returns the common elements from two sets.

These expressions perform the queries efficiently in a relational database environment using relational algebra operations.