DATABASE MANAGEMENT SYSTEM

Introduction:

Q-1: What do you know about databases? What are the advantages of DBMS. Discuss some drawbacks of using a file system to store data / List differences between a file processing system and DBMS. [MEC_C_01] [NIT_C_01] [FEC_C_01][STEC_Pre_01]

Answer:

A database management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to a database, contains information relevant to an enterprise.

There are several advantages of using a Database Management System (DBMS):

1. 💾 **Data Security:** DBMS includes built-in security features such as access control, user authentication, and encryption to protect sensitive data from unauthorized.
2. 🚀 **Data Integrity:** DBMS ensures data integrity by enforcing constraints and rules on data entry and modification. This helps to maintain accurate and reliable data.
3. ⚡ **Data Consistency**: DBMS maintains data consistency by providing techniques such as transactions and locking mechanisms. This ensures that data remains consistent even when multiple users are accessing or modifying it concurrently.
4. 🦹 **Data Accessibility:** DBMS allows multiple users to access and retrieve data simultaneously, providing a high level of concurrency. It also supports complex queries and provides efficient data retrieval mechanisms.
5. 🔄 **Data Sharing and Collaboration:** DBMS enables data sharing and collaboration among multiple users or applications. It allows users to access and modify data concurrently, facilitating teamwork and decision-making.
6. 📉 **Data Redundancy Reduction:** DBMS minimizes data redundancy by storing data in a structured and organized manner. This reduces storage space requirements and improves data maintenance.
7. 📝 **Data Independence:** DBMS provides a layer of abstraction between the physical storage and the applications accessing the data. This allows applications to remain unaffected by changes in the underlying data structure.
8. 🔍 **Data Retrieval and Analysis:** DBMS supports efficient search and retrieval operations, making it easier to extract specific information from large datasets. It also provides capabilities for data analysis and reporting.

Ordinary file system has a number of major drawbacks:

1. Data redundancy and inconsistency - Multiple file formats, duplication of information in different files

2. Difficulty in accessing data - Need to write a new program to carry out each new task

3. Data isolation - Multiple files and formats

4. Integrity problems - Integrity constraints (e.g. account balance > 0) become part of program code rather than being stated explicitly - Hard to add new constraints or change existing ones

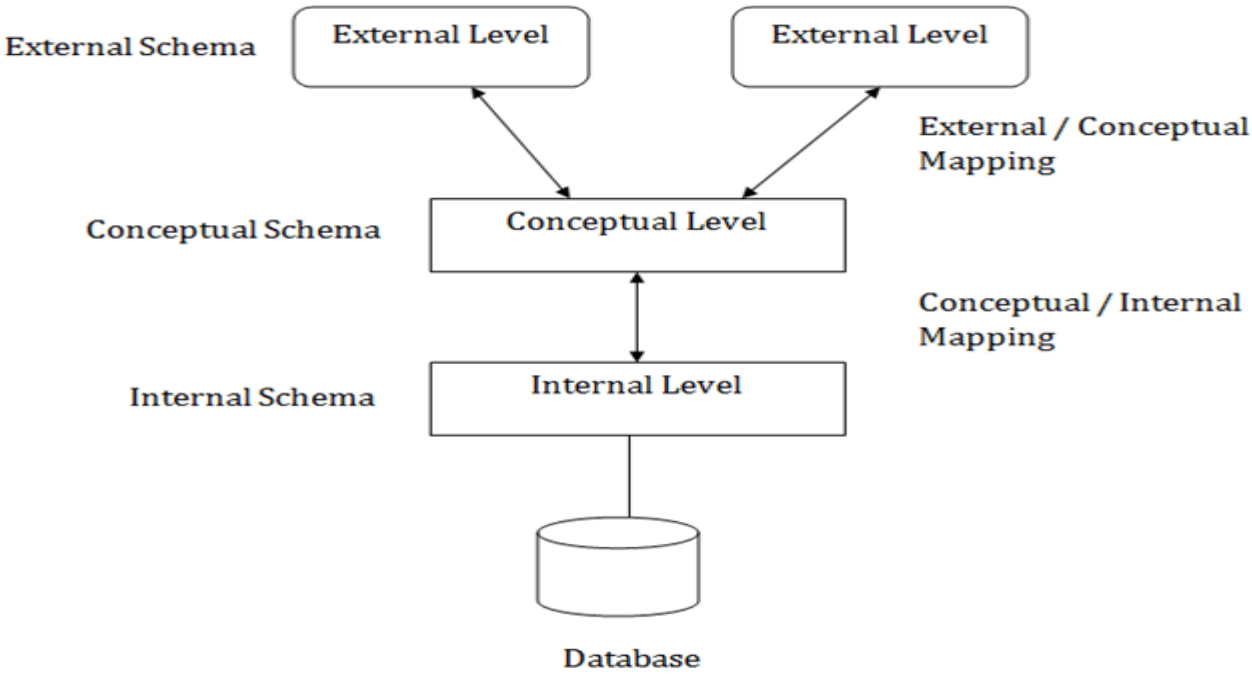
5. Atomicity problems - Failures may leave database in an inconsistent state with partial updates carried out. E.g., transfer of funds from one account to another should either complete or not happen at all

6. Security problems - Not every user of the database system should be able to access all the data. Hard to provide user access to some data, but not all .

Q-02: What is schema. Explain the three-schema architecture in DBMS? [STEC_C_01]

Answer:

Schema in the context of a database refers to the overall structure and organization of that database. It defines how data is organized and how different entities (tables) and their relationships are represented.



The three-schema architecture, also known as the ANSI-SPARC architecture, is a framework for designing and developing database systems. It consists of three levels:

1. **External Schema/View Schema:** This schema represents the way the data is viewed or perceived by individual users or specific applications. It allows different users or applications to have different views of the same underlying data, tailored to their specific requirements.
2. **Logical Schema (Conceptual Schema):** The conceptual schema represents the overall logical structure of the entire database. It defines the entities, their attributes, and the relationships between them at a high level. The conceptual schema acts as a bridge between the external schemas and the physical schema.
3. **Physical Schema (Internal Schema)**: The physical schema describes how the data is actually stored and accessed at the physical level of the database. It specifies details such as disk storage structures, indexing techniques, and access paths.

Q-03: How Foreign key maintains the references integrity for insertion, deletion and Updation? [STEC_C_01]

Answer: 🔑

Foreign keys play a crucial role in maintaining reference integrity during data manipulation operations like insertion, deletion, and updating in a database. Here's how foreign keys achieve this:

Insertion:

Invalid References: When inserting a new row into a table that has a foreign key, the foreign key ensures that the value being inserted matches an existing value in the referenced table's primary key. If no matching value is found, the insertion will be rejected, preventing the creation of orphaned records.

Deletion:

Protects Dependent Data: Upon deleting a row from the parent table, the DBMS defined foreign key constraints, considering:

- No Action: The deletion is prohibited if corresponding child rows exist, safeguarding data integrity.
- Cascade Delete: Automatically removes dependent rows in the child table, maintaining referential consistency.
- Set Null: Foreign key values in child rows are set to NULL, indicating the absence of a reference.
- Set Default: Foreign key values are assigned a predefined default value.

Updation:

- Ensures Consistent References: When updating a primary key value in the parent table, the DBMS propagates the change to corresponding foreign key values in the child table, ensuring alignment. This prevents orphaned rows.
- Restricts Violations: The DBMS may preclude updates that would result in invalid foreign key references in the child table, upholding integrity.

Q-04:What is Relation Algebra? Describe the Projection, Selection, Cross product and Set Differences in Relational Algebra. [STEC_C_01]

Answer:

Relation algebra is a mathematical framework used to manipulate and query relational databases. It defines a set of operations that can be performed on relations (tables) to retrieve specific data or derive new relations based on certain conditions. Here are brief explanations of four fundamental operations in relation algebra:

1. Projection (π):

- Selects specific columns from a relation.
- Syntax: π_A1, A2..., An(R)
- Example: π_Name, City(Customers) selects the "Name" and "City" columns from the "Customers" table.

2. Selection (σ):

- Filters rows based on a specific condition.
- Syntax: σ_predicate(R)
- Example: σ_Age > 25(Customers) selects customers older than 25 from the "Customers" table.

3. Cross Product (×):

- Creates a new relation by combining all rows from two relations.
- Syntax: $R_1 \times R_2$
- Example: Customers × Orders creates a relation with all combinations of customers and their orders.

4. Set Differences:

- These operators perform comparisons between relations:

  Union (∪): Combines rows from two relations, eliminating duplicates.

  - Syntax: $R_1 \cup R_2$

  Difference (-): Returns rows present in one relation but not the other.

  - Syntax: $R_1 - R_2$

  Intersection (∩): Returns rows present in both relations.

  - Syntax: $R_1 \cap R_2$

Q-05:What is Natural Join? Explain the natural join procedure with any examples. [STEC_C_01]

Answer:

The natural join is an operation in relational algebra that combines two relations based on their common attributes (columns). It matches tuples from both relations that have the same values in the common attributes and produces a new relation that includes all attributes from both relations.

The natural join procedure involves the following steps:

1. Find common attributes: Identify the attributes that exist in both relations and define them as the join attributes.
2. Match tuples: Compare the values of the join attributes in both relations. Tuples with matching values are combined to form the resulting relation.
3. Combine attributes: Include all attributes from both relations in the resulting relation. The join attributes are included only once, even if they appear in both relations.

Here's an example to illustrate the natural join procedure:

Consider two relations, "Employees" and "Departments":

Employees:

EmployeeID	Name	DepartmentID
1	Alice	101
2	Bob	102
3	Charlie	103
4	David	101

Departments:

DepartmentID	DepartmentName
101	HR
102	Sales
103	Marketing

To perform a natural join between these two relations based on the "DepartmentID" attribute:

1. Identify the common attribute: Both "Employees" and "Departments" have the "DepartmentID" attribute.
2. Match tuples: Compare the values of the "DepartmentID" attribute in both relations. Tuples with matching values are combined.

The resulting relation would be:

EmployeeID	Name	DepartmentID	DepartmentName
1	Alice	101	HR
2	Bob	102	Sales
3	Charlie	103	Marketing
4	David	101	HR

In this example, the resulting relation includes all attributes from both "Employees" and "Departments." It combines the tuples where the "DepartmentID" values match in both relations, providing information about employees and their respective departments.

The natural join is a useful operation for combining related data from multiple relations based on common attributes, enabling further analysis and data retrieval.

Q-06: "DDL" and "DML" are two separate languages in practices"-Justify. [NIT_C_01] [STEC_C_02]

Answer:

Data-Definition Language (DDL):

• A database schema is specified by a set of definitions expressed by a special language called a data-definition language.

Example DDL statements or DDL Commands:

1. **CREATE:** Used to create new database objects, such as tables, indexes, views, or schemas. It specifies the structure and constraints for the new object.
2. **ALTER:** Allows you to modify the structure of an existing database object. You can add, modify, or delete columns, change data types, and more.
3. **DROP:** Used to delete existing database objects, such as tables, views, or indexes. This permanently removes the object and its data.
4. **TRUNCATE:** Removes all rows from a table, effectively emptying it, but leaves the table structure intact. It's a faster way to remove all data compared to DELETE.
5. **RENAME:** Renames an existing database object, such as a table or column, to a new name.

Data-Manipulation Language (DML):

• DML is a language that enables users to access or manipulate data as organized by appropriate data model.

DML Commands:

1. **SELECT:** Used to retrieve data from one or more tables. It allows you to query the database and retrieve specific information based on specified criteria.
2. **INSERT:** Adds new data to a table. You can insert one or more rows into a table, providing values for each column.
3. **UPDATE:** Modifies existing data in a table. It allows you to change the values of specific rows and columns.
4. **DELETE:** Removes data from a table, typically by specifying a condition that identifies which rows should be deleted.

Q-07: Write down the difference between Drop, Truncate, and Delete. [STEC_C_02]

Answer:

	DROP	DELETE	TRUNCATE
Definition	It completely removes the table from the database.	It removes one or more records from the table.	It removes all the rows from the existing table
Type of Command	It is a DDL command	It is a DML command	It is a DDL command
Syntax	DROP TABLE table_name;	DELETE FROM tble_nameWHERE conditions;	TRUNCATE TABLE table_name;
Memory Management	It completely removes the allocated space for the table from memory.	It doesn't free the allocated space of the table.	It doesn't free the allocated space of the table.
Effect on Table	Removes the entire table structure.	Doesn't affect the table structure	Doesn't affect the table structure
Speed and Performance	It is faster than DELETE but slower than TRUNCATE as it firstly deletes the rows and then the table from the database.	It is slower than the DROP and TRUNCATE commands as it deletes one row at a time based on the specified conditions.	It is faster than both the DELETE and DROP commands as it deletes all the records at a time without any condition.
Use with WHERE clause	Not applicable as it operates on the entire table	Can be used	It can't be used as it is applicable to the entire table

Q-08: Write short notes on Candidate key and Composite key. [STEC_C_02]

Answer:

Candidate Key: The minimal set of attributes that can uniquely identify a tuple is known as a candidate key.

Note: Candidate key attributes can contain null values, but will not consider same value.

Reg_id	Sec	name	age	Email_id	Phone_no
100	A	Farhan	20		
101	A	Hossan	22		
102	B	Rabbi	20		
103	B	Nayon	21		
104	B	Hossan	21		
105	A	Rana	23		

Candidate Key:

- Uniquely identifies each row in a table.
- A candidate key must be unique but it can be null.
- Can be single attribute or combination of attributes.
- Multiple candidate keys may exist per table.
- Not chosen as the primary key by default.

Composite Key:

- A candidate key with two or more attributes.
- Individual attributes may not be unique, but their combination is.
- Often used when no single attribute is unique.
- Can be the primary key if chosen.
- A composite key is a primary key composed of two or more columns.

Q-09:Write an appropriate query for the following table named Employee. [STEC_C_02]

Employee table:

Emp_id	Emp Name	Emp Address	Emp Blood_Group	Salary	Dept_id
1001	Rahim	Dhaka	A+	500	101
1002	Karim	Dhaka	B+	400	102
1003	Ramim	Dhaka	AB+	300	103
1004	Sarim	Dhaka	O+	600	102
1005	Robin	Dhaka	AB-	1000	101

i) Write a query to create an Employee table as given above. Consider Emp_id as the primary key and Dept_id as the foreign key.

ii) Write a query to insert values according to the table given above.

Answer:

i) Query to create the Employee table:

```
CREATE TABLE Employee (
   Emp_id INT PRIMARY KEY,
   Emp_Name VARCHAR(50),
   Emp_Address VARCHAR(100),
   Emp_Blood_Group VARCHAR(3),
   Salary INT,
   Dept_id INT,
   FOREIGN KEY (Dept_id) REFERENCES Department (Dept_id)
);
```

In the above query:

- **Emp_id** is the primary key.
- **Dept_id** is a foreign key referencing the **Dept_id** in the **Department** table.

ii) Query to insert values into the Employee table:

```
INSERT INTO Employee (Emp_id, Emp_Name, Emp_Address, Emp_Blood_Group, Salary, Dept_id)
VALUES
  (1001, 'Rahim', 'Dhaka', 'A+', 500, 101),
  (1002, 'Karim', 'Dhaka', 'B+', 400, 102),
  (1003, 'Ramim', 'Dhaka', 'AB+', 300, 103),
  (1004, 'Sarim', 'Dhaka', 'O+', 600, 102),
  (1005, 'Robin', 'Dhaka', 'AB-', 1000, 101);
```

In the above query:

- Each **VALUES** clause represents a row to be inserted into the **Employee** table.
- Make sure to provide values in the same order as the columns in the table.

Q-10: Write an appropriate query for the above table named Employee and the following table named Department. [STEC_C_02]

Department table:

Dept_id	Dept Name	Dept_Location
101	CSE	First Floor
102	EEE	Third Floor
103	Civil	Fifth Floor
104	Architecture	Sixth Floor

i) Write a query to find the maximum, minimum, and average salary from the Employee table.

ii) Write a query to find the Employee whose names start with "R" and end with "im"?

iii) Write a query to change the department location for Architecture to the seventh floor.

iv) Write a query to add a column named Age to the Employee table.

v) Write a query to display Emp_id, Dept_id, Emp Name, Dept Name, and Salary.

Answer:

i) Query to find the maximum, minimum, and average salary from the Employee table:

```
SELECT MAX(Salary) AS Max_Salary,
       MIN(Salary) AS Min_Salary,
       AVG(Salary) AS Avg_Salary
FROM Employee;
```

In the above query, the `SELECT` statement is used to retrieve the maximum (`MAX`), minimum (`MIN`), and average (`AVG`) salary from the Employee table. The `AS` keyword is used to assign aliases to the result columns for better readability.

ii) Query to find the Employee whose names start with "R" and end with "im":

```
SELECT *
FROM Employee
WHERE Emp_Name LIKE 'R%im';
```

In the above query, the `SELECT` statement is used to retrieve all columns (`*`) from the Employee table. The `WHERE` clause is used to filter the rows based on the condition specified with the `LIKE` operator. The `%` wildcard is used to match any number of characters, so the condition `'R%im'` will match the names that start with "R" and end with "im".

iii) Query to change the department location for Architecture to the seventh floor:

```
UPDATE Department
SET Dept_Location = 'Seventh Floor'
WHERE Dept_Name = 'Architecture';
```

In the above query, the `UPDATE` statement is used to modify the Department table. The `SET` clause is used to specify the column and its new value, and the `WHERE` clause is used to identify the row to be updated based on the condition specified with the `Dept_Name`.

iv) Query to add a column named Age to the Employee table:

```
ALTER TABLE Employee
ADD Age INT;
```

In the above query, the `ALTER TABLE` statement is used to modify the structure of the Employee table. The `ADD` keyword is used to add a new column named "Age" with the specified data type (`INT`).

v) Query to display Emp_id, Dept_id, Emp Name, Dept Name, and Salary:

```
SELECT e.Emp_id, e.Dept_id, e.Emp_Name, d.Dept_Name, e.Salary
FROM Employee e
JOIN Department d ON e.Dept_id = d.Dept_id;
```

In the above query, the `SELECT` statement is used to retrieve Emp_id, Dept_id, Emp Name, Dept Name, and Salary from the Employee and Department tables. The `JOIN` keyword is used to combine the rows from both tables based on the matching Dept_id values.

Q-11: Define schema with example. Explain the differences between physical and logical data independences. [FEC_C_01]

Answer: Schema: In database management, a schema refers to the logical structure or blueprint that defines the organization and layout of a database.

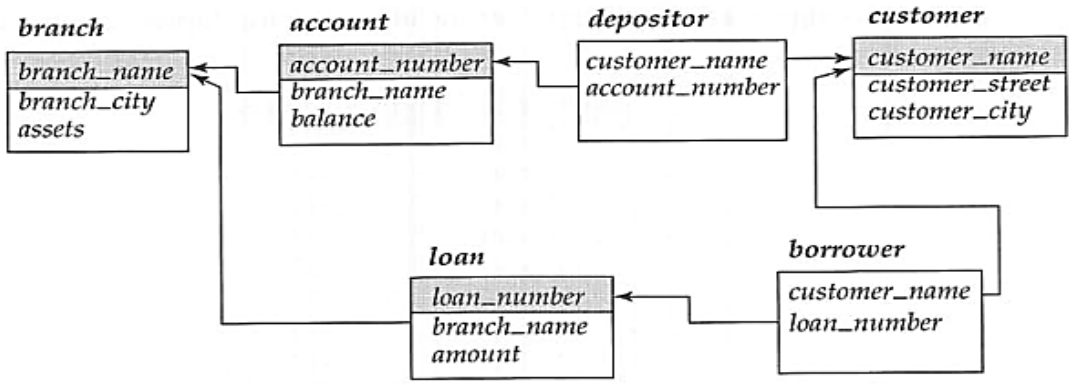


Figure 2.8 Schema diagram for the banking enterprise.

Aspect	Physical Data Independence	Logical Data Independence
Focus	Deals with the storage of data	Deals with the structure or changing the data definition
Data Retrieval	Retrieval is easy	Retrieval is difficult
Ease of Achievement	Relatively easy to achieve compared to logical data independence	Difficult to achieve logical data independence
Impact on Applications	Application programs are not changed if there is a change in the physical level	Changes are needed in the application program if new fields are added or deleted from the database
Schema Level	Deals with the internal schema	Deals with conceptual schema
Examples	Hashing algorithms, storage devices, etc.	Adding, deleting, or modifying a new attribute

Q-12: What do you mean database administrators? What are the (five main) functions of database administrators? [FEC_C_01]

Answer:

Database administrators (DBAs) are professionals responsible for managing and maintaining databases.

Here are the five main functions of database administrators:

1 Database Design: DBAs participate in the design process of a database, including determining the data structure, defining relationships between tables, and optimizing performance.

2 Database Installation and Configuration: DBAs install database management systems (DBMS) and configure them according to the organization's requirements and industry best practices.

3 Performance Monitoring and Tuning: DBAs monitor database performance, identify bottlenecks, and optimize the system to ensure efficient and smooth operation. This involves tasks like indexing, query optimization, and resource allocation.

4 Backup and Recovery: DBAs establish backup and recovery strategies to protect the database from data loss and ensure disaster recovery. They schedule regular backups, perform data restoration when needed, and implement security measures to safeguard data integrity.

5 Security Administration: DBAs are responsible for enforcing data security measures. They manage user access, permissions, and roles, implement encryption and authentication mechanisms, and oversee compliance with data protection regulations.

Overall, database administrators play a crucial role in designing, maintaining, securing, and optimizing databases to ensure their performance, reliability, and data integrity.

Q-13: Define referential integrity constrains. Briefly explain different mapping cardinalities with mapping diagram. [FEC_C_01]

Answer: Referential integrity constraints ensure that data relationships between tables in a database are maintained and accurate. They prevent orphaned records and enforce valid connections between tables. 🔁 🔒

Here are the commonly used mapping cardinalities cardinalities with mapping diagram:

1. One-to-One (1:1): In this relationship, one instance of an entity is associated with exactly one instance of another entity. It is represented by a straight line between the entities.

2. One-to-Many (1:N): This relationship indicates that one instance of an entity is associated with multiple instances of another entity. It is represented by a straight line with a crow's foot symbol on the "many" side.

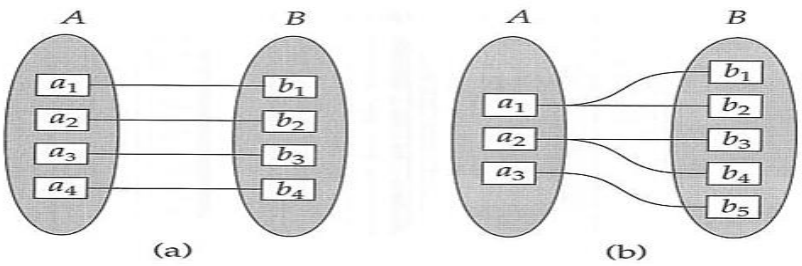


Figure 6.5 Mapping cardinalities. (a) One-to-one. (b) One-to-many.

3. Many-to-One (N:1): This relationship shows that multiple instances of an entity are associated with exactly one instance of another entity. It is represented by a crow's foot symbol on the "one" side of the line.

4. Many-to-Many (N:N): In this relationship, multiple instances of an entity are associated with multiple instances of another entity. It is represented by a crow's foot symbol on both sides of the line. However, a many-to-many relationship is typically implemented using an intermediary table in the database.

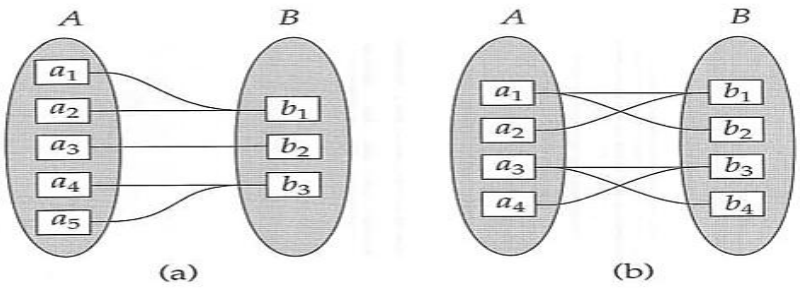


Figure 6.6 Mapping cardinalities. (a) Many-to-one. (b) Many-to-many.

Q-14: What do you mean by functional dependency? Write Armstrong's axioms property of functional dependency. [FEC_C_01]

Answer:

Functional dependency in the context of databases is a relationship between two sets of attributes within a database table. It describes the dependency of one set of attributes (dependent attributes) on another set of attributes (determinant attributes).

Armstrong's axioms are a set of properties that hold true for functional dependencies. These properties help in determining and proving other functional dependencies based on given functional dependencies. Armstrong's axioms consist of three properties:

Reflexivity: If Y is a subset of X, then X → Y (X functionally determines Y)

If X is a set of attributes and Y is a subset of X, then X → Y holds true. In other words, any set of attributes is functionally dependent on its own subset.

Augmentation: If X → Y, then XZ → YZ for any Z (adding the same attributes on both sides).

If X → Y holds true and Z is a set of attributes, then XZ → YZ also holds true. This property states that if Y is functionally dependent on X, then it is also functionally dependent on any superkey that includes X.

Transitivity: If X → Y and Y → Z, then X → Z (transitive dependency).

If X→ Y holds true and Y → Z holds true, then X → Z also holds true. This property states that if Y is functionally dependent on X, and Z is functionally dependent on Y, then Z is functionally dependent on X.

These axioms form the basis for establishing and proving other functional dependencies through a process known as closure calculation. By applying the axioms repeatedly, we can determine additional functional dependencies that exist within a given set of attributes.

Q-15:Explain the differences between a super key and a candidate key: [FEC_C_01]

Answer:

Aspect	Candidate Key	Super Key
Definition	A candidate key is a minimal set of attributes that can uniquely identify a record within a table.	A super key is a set of one or more attributes (columns) that can uniquely identify a record within a table.
Relation	While in a relation, number of candidate keys are less than number of super keys	In a relation, number of super keys are more than number of candidate keys
Impact	Candidate key's attributes can contain NULL values	Super key's attributes can also contain NULL values
Subset	Candidate Key is a proper subset of a super key.	Super Key is an attribute (or set of attributes) that is used to uniquely identifies all attributes in a relation.
Identifier	All candidate keys are super keys.	All super keys can't be candidate keys
Criteria	Various candidate keys together makes the criteria to select the primary keys.	Various super keys together makes the criteria to select the candidate keys

Q-16: Consider an insurance database that maintain the following information : [FEC_C_02]

*Person (**DriverNo**, Name, Address)*
*Car (**LicenseNo**, Model, Year)*
*Accident (**ReportNo**, **Date**, Location)*
*Owns (**DriverNo**, **LicenseNo**)*
*Participated (**DriverNo**, **LicenseNo**, **ReportNo**, **DamageAmount**)*
Write down the SQL command to find the following queries:

i. Find the total number of people whose cars were involved in accident.
ii. Find the number of accident in which the belongs to “Mr. X”.
iii. Delete the “CAR1” car belongs to “Mr. X”.
iv. Add new accident report for the “CAR2” car belongs to “Mr. Y”.

Here are the SQL commands for the given queries:

i. Find the total number of people whose cars were involved in accidents:

SQL

```sql
SELECT COUNT(DISTINCT Person.DriverNo)
FROM Person
JOIN Participated ON Person.DriverNo = Participated.DriverNo;
```

Use code with caution. Learn more

ii. Find the number of accidents in which the cars belong to "Mr. X":

SQL

```sql
SELECT COUNT(*)
FROM Accident
JOIN Participated ON Accident.ReportNo = Participated.ReportNo
JOIN Person ON Participated.DriverNo = Person.DriverNo
WHERE Person.Name = 'Mr. X';
```

Use code with caution. Learn more

iii. Delete the "CAR1" car belonging to "Mr. X":

SQL

```sql
DELETE FROM Car
WHERE LicenseNo = 'CAR1'
AND LicenseNo IN (
    SELECT LicenseNo
    FROM Owns
    JOIN Person ON Owns.DriverNo = Person.DriverNo
    WHERE Person.Name = 'Mr. X'
);
```

Use code with caution. Learn more

iv. Add a new accident report for the "CAR2" car belonging to "Mr. Y":

SQL

```sql
INSERT INTO Accident (ReportNo, Date, Location)
VALUES ('AR20240126', '2024-01-26', 'Some Location');  -- Replace with actual values

INSERT INTO Participated (DriverNo, LicenseNo, ReportNo, DamageAmount)
SELECT DriverNo, LicenseNo, 'AR20240126', 3000  -- Replace with actual values
FROM Owns
JOIN Person ON Owns.DriverNo = Person.DriverNo
WHERE LicenseNo = 'CAR2' AND Person.Name = 'Mr. Y';
```

Use code with caution. Learn more

Q-17: What is the primary purpose of using views in a database? Explain the concept of an updatable view and the conditions that must be met to enable data manipulation operations through a view. [FEC_C_02]

Answer:

The primary purpose of using views in a database is to provide a virtual representation of the data from one or more tables. Views act as predefined queries that allow users to retrieve specific data without exposing the underlying structure of the database. They offer the following benefits:

Here are some key benefits of using views:

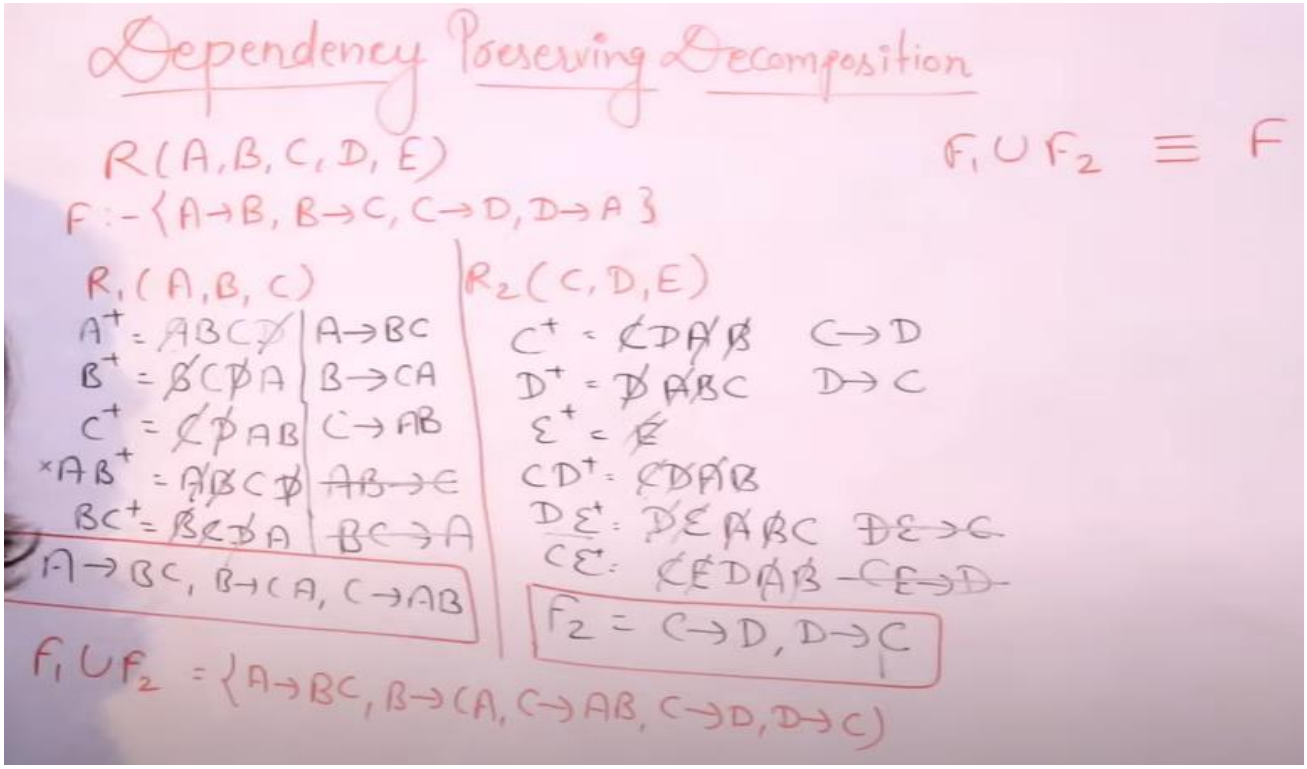
- Data Abstraction: Views simplify complex queries by encapsulating them into a single named entity. Users don't need to understand the underlying table structure or intricate joins, just the view's definition.
- Security: Views can restrict access to sensitive data by limiting which columns or rows are visible to specific users. This enhances data security by granting access based on need rather than full table permissions.
- Data Presentation: Views can present data in a way that aligns with different user needs or application requirements. For example, a sales team might have a view showing current customer orders, while managers might see aggregated sales data by region.
- Standardization: Frequently used complex queries can be stored as views, ensuring consistency and reducing code duplication across applications or users.
- Performance Optimization: In some cases, views can optimize query performance by pre-joining tables or filtering data upfront, especially for frequently accessed data subsets.

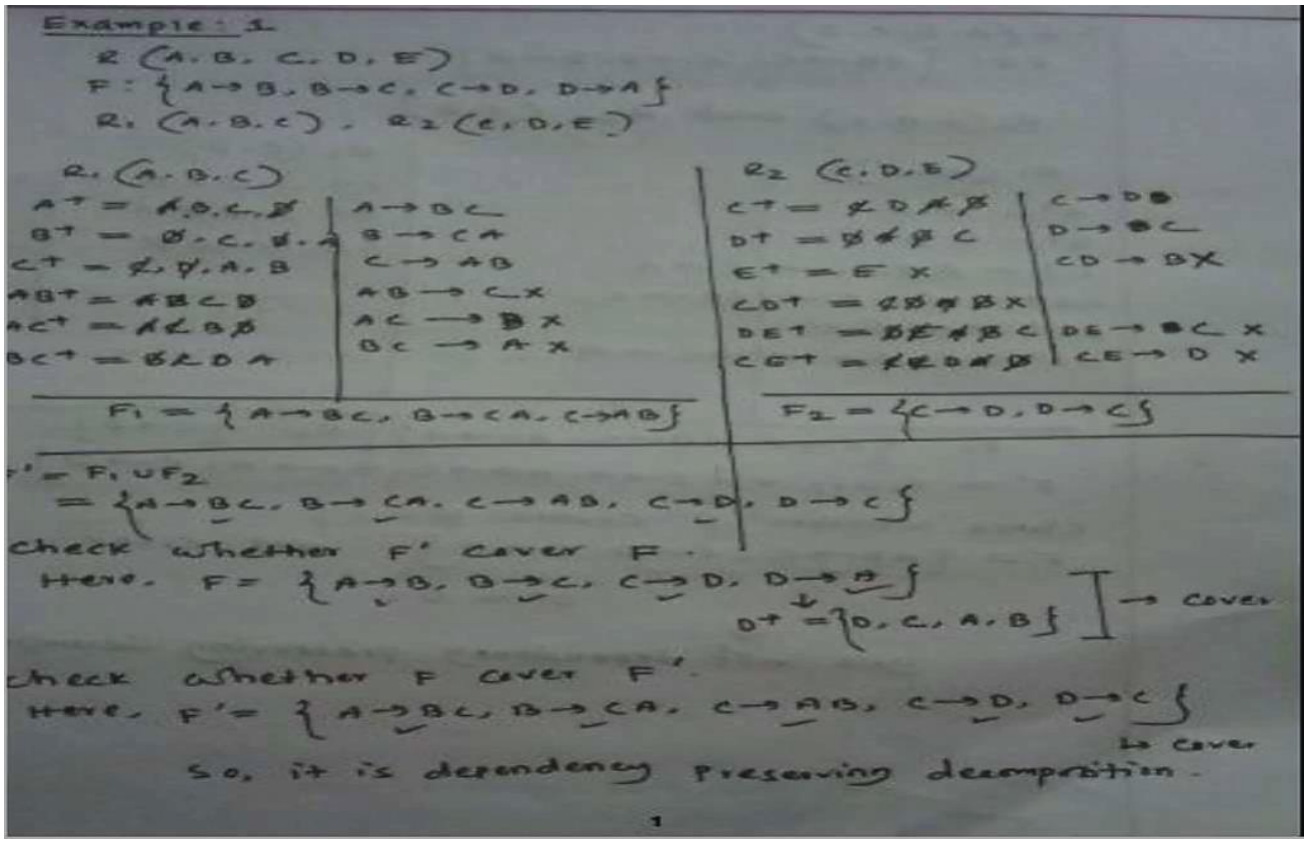
However, not all views allow data manipulation:

- Updatable Views: These permit INSERT, UPDATE, and DELETE operations through the view, modifying the underlying tables. Specific conditions must be met for a view to be updatable:
    - Simple SELECT statement: The view definition must be a single SELECT statement without complex logic like aggregations or GROUP BY clauses.
    - Deterministic values: Any calculations or expressions in the view must produce deterministic results (returning the same output for the same input every time).
    - Updatable base tables: The underlying tables referenced by the view must be updatable with appropriate permissions.
    - No joins to non-updatable tables: If the view joins to tables that are not updatable, it cannot support modifications through the view.
- Read-only Views: These views cannot be used for data manipulation and are primarily for data presentation and querying.

Q-18:Suppose that we decompose the schema R = (A,B,C,D,E) into R1(A,B,C) and R2(C,D,E). Check whether this decomposition is dependency preserving or not where the set F of functional dependencies holds: A->B, B->C, C->D, D->A. [FEC_C_01]

Answer:





Q-19:Define canonical cover. Suppose R = (A,B,C) and set of functional dependencies F as follows: A->BC, B->C, A->B, AB->C. Find out the canonical cover. [FEC_C_01]

Answer:

- A canonical cover of F is a **minimal set** of functional dependencies equivalent to F, **having no redundant dependencies** or **redundant parts of dependencies.**

$R = (A, B, C)$
$F = \{A \rightarrow BC$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C\}$

Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
- Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

A is extraneous in $AB \rightarrow C$
- Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
  - Yes: in fact, $B \rightarrow C$ is already present!
- Set is now $\{A \rightarrow BC, B \rightarrow C\}$

C is extraneous in $A \rightarrow BC$
- Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
  - Yes: using transitivity on $A \rightarrow B$ and $B \rightarrow C$.
    - Can use attribute closure of A in more complex cases

The canonical cover is: $A \rightarrow B$
$B \rightarrow C$

Q-20: What do you mean by data abstraction? What are the different levels of data abstraction in DBMS. [STEC_Pre_01]

Answer: Data abstraction refers to the process of simplifying complex data structures by representing them in a more understandable and manageable way. It involves hiding the unnecessary details and exposing only relevant information to users and applications.

1. Physical Level:

- ❖ The lowest level of abstraction.
- ❖ Describes how data is actually stored on the physical storage media such as hard drives or memory.
- ❖ Deals with details like disk blocks, file systems, and data storage formats.
- ❖ Example: Representation of data in binary format using bits, bytes, and disk blocks.

2. Logical Level:

- ❖ The next higher level of abstraction.
- ❖ Focuses on the structure and organization of data without considering the physical storage aspect.
- ❖ Concerned with defining entities, attributes, relationships, and constraints.
- ❖ Example: Tables, relationships, and constraints defined in a relational database.

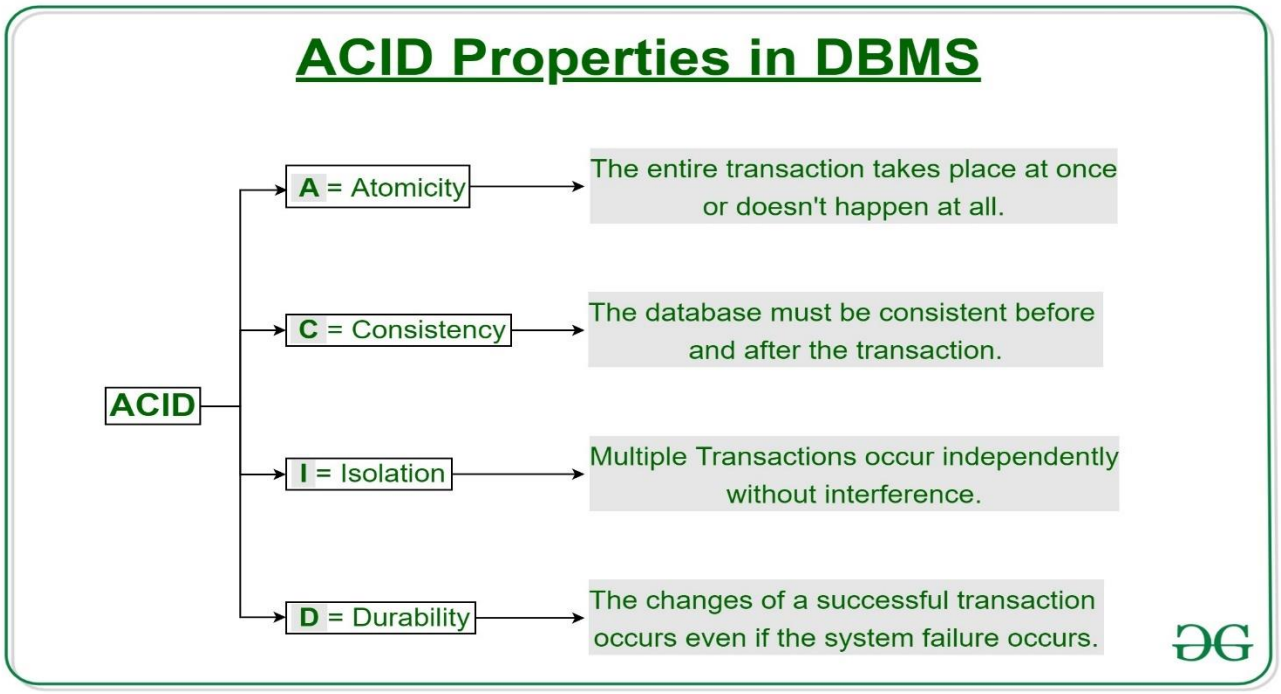
3. View Level:

- ❖ The highest level of abstraction.
- ❖ Represents a specific subset of data that is relevant to a user or a particular application.
- ❖ Provides a customized view of the data according to the user's requirements, hiding unnecessary details.

These levels of data abstraction in DBMS enable separation of concerns, data independence, and provide flexibility in managing and accessing data. They allow users to work with data at different levels of complexity, from low-level storage details to high-level conceptual views.

Q-21:Explain the concept of ACID properties in DBMS? [STEC_Pre_01]

Answer:



1. Atomicity: It means that either all the operations within a transaction are executed successfully, or none of them are executed at all. If any part of the transaction fails, the entire transaction is rolled back, and the database is restored to its state before the transaction began. This ensures data integrity and prevents the database from being left in an inconsistent state.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

2. Consistency: Consistency ensures that a transaction brings the database from one valid state to another valid state. It means that the data must satisfy all the integrity constraints defined in the

database schema after the completion of a transaction. The consistency constraint ensures that the database remains in a consistent state even if transactions are executed concurrently.

Total **before T** occurs = **500 + 200 = 700.**
Total **after T occurs** = **400 + 300 = 700.**

3. Isolation: Isolation ensures that each transaction is executed in isolation from other concurrent transactions. It means that the intermediate state of a transaction is not visible to other transactions until it is committed. This property prevents interference and maintains data integrity by ensuring that concurrent transactions do not interfere with each other's results. Isolation is achieved through concurrency control mechanisms such as locking, serializability, and isolation levels.

T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write (Y)	Read (X) Read (Y) Z: = X + Y Write (Z)

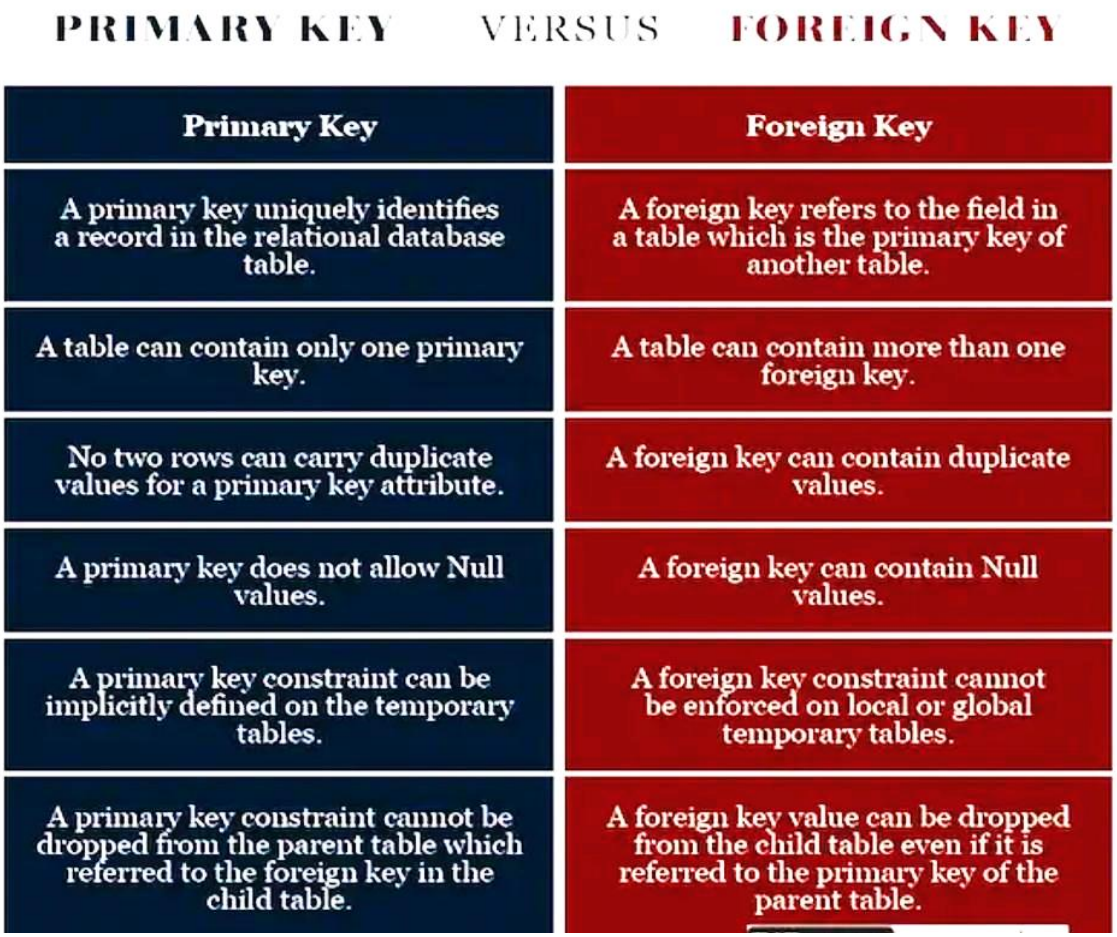
4. Durability: Durability ensures that once a transaction is committed, its effects are permanent and survive any subsequent system failures, such as power outages or crashes. Once the DBMS sends a confirmation of a successful transaction commit, it guarantees that the changes made by the transaction are permanently stored in the database and will not be lost even in the presence of failures.

Real-world examples of ACID properties:

- Transferring money between bank accounts: This action typically involves multiple operations like debiting one account and crediting another. Atomicity ensures the entire transfer happens or none at all, avoiding partial transfers.
- Booking a flight ticket: This includes checking seat availability, reserving the seat, and updating the inventory. Isolation ensures no two passengers can book the same seat simultaneously.
- Saving a product order: This involves updating cart items, customer information, and inventory. Durability guarantees the order is permanently saved even if the system crashes before completion.

Q-22: Differences between Primary Key and Foreign Key. [MEC_C_01] [STEC_Pre_01]

Answer:



Q-23:Define the terms data,information,database and DBMS. [FEC_C_01(Pre)]

Answer: Here are simplified definitions of the terms:

1. Data: Data refers to raw facts, figures, or symbols that represent information. It can be in the form of numbers, text, images, or any other format. Data has no meaning on its own and requires interpretation to be useful.

2. Information: Information is processed and organized data that has meaning and context. It is obtained by interpreting and analyzing data, providing it with relevance and value. Information can be used to make decisions or gain knowledge.

3. Database: A database is a structured collection of organized data. It is designed to store, manage, and retrieve information efficiently. Databases consist of tables, which are used to organize data into rows (records) and columns (attributes).

4. DBMS (Database Management System): DBMS is a software system that enables the management and manipulation of databases. It provides a set of tools and functionalities to create, modify, store, and retrieve data from a database.

Q-24: Write the differences between instance and schema. Explain the Database system with proper diagram. [FEC_C_01(Pre)]

Answer:

Schema: In database management, a schema refers to the logical structure or blueprint that defines the organization and layout of a database.

Differences between an instance and a schema:

Parameters	Schema in DBMS	Instance in DBMS
Meaning	Schema refers to the overall description of any given database.	Instance basically refers to a collection of data and information that the database stores at any particular moment.
Alterations	The schema remains the same for the entire database as a whole.	One can change the instances of data and information in a database using updation, deletion, and addition.
Frequency of Change	It does not change very frequently.	It changes very frequently.
Uses	We use Schema for defining the basic structure of any given database. It defines how the available needs to get stored.	We use Instance for referring to a set of information at any given instance/ time.

Instance:

- Instance refers to the actual data stored in a database at a specific moment.
- It represents the real records and values that exist in the database.
- Example: Database Instance:

    -> Table: Employees

    -> Record 1: {ID: 101, Name: John, Dept: IT}

    -> Record 2: {ID: 102, Name: Jane, Dept: HR}

Schema:

- Schema defines the structure and organization of a database.
- It describes the blueprint or plan for how the data should be stored and organized.

- Example: Database Schema:

    -> *Table: Employees*

    -> *Columns: ID, Name, Dept*

In summary, an instance represents the actual data in the database, while a schema defines the structure and organization of the database.

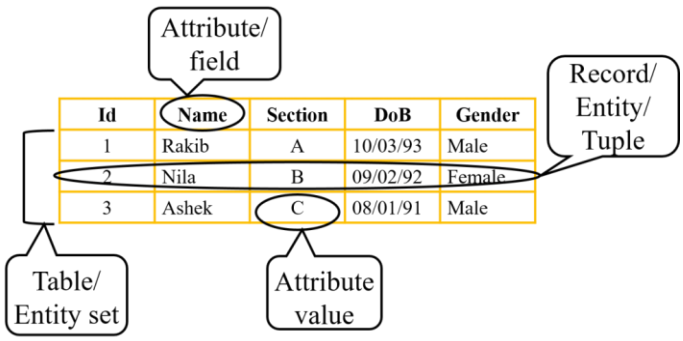
Q-25: What is an entity-relationship diagram? Draw an ER diagram. Discuss about different languages. [FEC_C_01(Pre)]

Answer:

An entity-relationship (ER) diagram is a visual representation of the entities (objects or concepts) within a system or domain, their attributes, and the relationships between them. It is a widely used model in database design to represent the structure of a database and its entities.

Components in an ER diagram:

1 Entities 2 Attributes 3 Relationships



ER diagram:

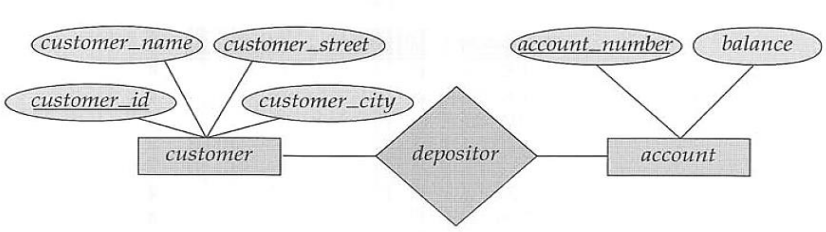


Figure 1.3 A sample E-R diagram.

Different languages used to create ER diagrams:

1 **Chen Notation:** Developed by Peter Chen, it uses rectangles to represent entities, diamonds for relationships, and lines to connect them. Attributes are typically shown inside the entity rectangles.

2 **Crow's Foot Notation:** Also known as Information Engineering notation, it uses a crow's foot symbol (three lines) to represent a many-side of a relationship. The entities and attributes are similar to Chen notation.

3 **Unified Modeling Language (UML):** UML can be used to create ER diagrams as well. It provides a wider range of symbols and notations for different types of relationships and constraints.

Q-26: Define Key. Explain the following terms in a relational database: [FEC_C_01(Pre)]

- Candidate key
- Super key
- Primary key
- Foreign key

Answer:

A key is a special field or combination of fields in a database table that helps uniquely identify each record. It is used to establish relationships between tables and ensure data integrity.

Candidate Key 🔑:

A candidate key is a set of one or more columns in a table that can uniquely identify each record. It means that the combination of values in the candidate key columns should be unique for every row in the table.

Super Key 🔑🔑:

A super key is a set of one or more columns that can uniquely identify each record in a table. It can include extra columns that are not necessary for uniqueness.

Primary Key 🆔:

A primary key is a special candidate key chosen as the main identifier for a record in a table. It must be unique and non-null and serves as a reference point for establishing relationships with other tables.

Example:

In the "Students" table, if we choose StudentID as the primary key, it will uniquely identify each student record. No two students can have the same StudentID.

Foreign Key 🔗:

A foreign key is a column or set of columns in a table that establishes a link to the primary key of another table. It ensures data integrity by enforcing relationships between tables.

Q-27: Why decomposition of a relation is required? [FEC_C_01(Pre)]

Answer:

Here are some key reasons why decomposition is necessary:

Data Organization: Decomposing a relation allows for the logical organization of data into smaller, more manageable parts.

Eliminating Redundancy: Redundancy refers to the duplication of data within a database. Decomposition helps eliminate or minimize redundancy by redistributing and structuring the data efficiently.

Data Dependencies: Decomposition is often done to ensure that data dependencies and relationships are properly represented and maintained.

Flexibility and Scalability: Decomposition provides flexibility and scalability in the database design. It allows for modifications, additions, or deletions of attributes, entities, or relationships without affecting the entire database structure.

Query Optimization: Decomposing a relation can improve query optimization.

Overall, decomposition of a relation is essential for optimizing data organization, improving efficiency, reducing redundancy, and maintaining data integrity and dependencies within a database. It contributes to a well-designed and scalable database system. 😊 🗂️

Q-28:Consider a relation R (A, B, C, D) with the functional dependencies A->B, B->C, C->D. We want to know whether the decomposition of R into R1 (A, B), R2 (B, C) and R3 (C, D) is lossless or lossy. [FEC_C_01(Pre)]

To determine whether the decomposition of relation R (A, B, C, D) into R1 (A, B), R2 (B, C), and R3 (C, D) is lossless or lossy, we can perform the lossless-join decomposition test.

The lossless-join decomposition test involves checking if the natural join of the decomposed relations can give us the original relation back without losing any information.

Let's perform the test:

1. Natural Join of R1 (A, B) and R2 (B, C):

   R1 ⋈ R2 (A, B, C) = R (A, B, C)

2. Natural Join of R2 (B, C) and R3 (C, D):

   R2 ⋈ R3 (B, C, D) = R (A, B, C, D)

If the natural join of decomposed relations (R1, R2, R3) produces the original relation R, the decomposition is lossless. And if the natural join does not give us R or implies additional dependencies, the decomposition is lossy.

In this case, since both join operations result in the original relation R, the decomposition of R into R1, R2, and R3 is lossless. 😀 ✅

Q-29: What is view? How can you create a view? Discuss with example. [FEC_C_01(Pre)]

Answer:

To create a view in a database, you can use the CREATE VIEW statement. A view is a virtual table derived from one or more underlying tables or views, and it does not store any physical data. Instead, it dynamically retrieves data from the underlying tables whenever the view is queried.

Here's an example to illustrate how to create a view:

Assume we have two tables: "Customers" and "Orders". The "Customers" table has columns: "CustomerID", "CustomerName", "Email", and "Phone". The "Orders" table has columns: "OrderID", "CustomerID" (foreign key referencing the Customers table), "OrderDate", and "TotalAmount".

We can create a view that combines information from both tables to display the customer's name, order ID, and total amount for each order. Let's call the view "CustomerOrderView".

Here's how you can create the view in SQL:

```
CREATE VIEW CustomerOrderView AS
SELECT Customers.CustomerName, Orders.OrderID, Orders.TotalAmount
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

In the above example, the view "CustomerOrderView" is created by selecting the required columns from both the "Customers" and "Orders" tables using an INNER JOIN on the "CustomerID" column.

Once the view is created, you can query it like a regular table:

```
SELECT * FROM CustomerOrderView;
```

This query will retrieve the customer's name, order ID, and total amount from the view, which in turn retrieves the data dynamically from the underlying tables.

Views provide a convenient way to abstract complex queries or present a subset of data to users, while hiding the underlying table structure and implementation details. They can also be used to enforce security restrictions, as you can grant or restrict access to specific views based on user privileges.

Q-30: Define multivalued dependency. Describe 1st and 2nd normal form. [FEC_C_01(Pre)]

Answer:

A multivalued dependency (MVD) is a concept in database normalization that describes a relationship between attributes in a table. It occurs when two or more sets of attributes are functionally dependent on a common set of attributes, but not on each other.

Employee_skills table:

Employee_ID	Skill_ID	Project_ID
001	S1	P1
001	S1	P2
002	S2	P1
002	S2	P3

In this example, Employee with ID 001 and Skill S1 can work on both projects P1 and P2. Similarly, Employee with ID 002 and Skill S2 can work on projects P1 and P3.

The MVD {Employee_ID, Skill_ID} ->> {Project_ID} ensures that if we have multiple employees with the same ID and Skill, the Project_ID can have multiple values, indicating that they can work on several projects.

Normalization aims to eliminate or reduce such dependencies to improve data integrity and eliminate data anomalies. The process of normalization leads to achieving different normal forms.

First Normal Form (1NF):

To satisfy the first normal form, a table must meet the following conditions:

1. Eliminate duplicate rows.

2. Each column should contain atomic (indivisible) values.

3. The order of rows and columns is irrelevant.

Second Normal Form (2NF):

A table is considered to be in the second normal form if it satisfies the following criteria:

1. It is in the first normal form.

2. All non-key attributes are fully functionally dependent on the whole primary key.

Q-31: Define extraneous attributes. Consider the following relation R (A, B, C) with the functional dependencies {A->B, C->B}. Compute the Boyce-Codd Normal Form (BCNF) of this relation. Does the relation contain any extraneous attributes? [FEC_C_01(Pre)]

Answer:

1. **Write down the given relation and functional dependencies:**
   $R(A, B, C)$ with functional dependencies $\{A \rightarrow B, C \rightarrow B\}$.
2. **Check for candidate keys:**
   Identify candidate keys using the given functional dependencies. In this case, the candidate key is AC because both A and C determine B, and no proper subset of AC has this property.
3. **Decompose the relation:**
   Decompose the relation into smaller relations, each containing a candidate key.

$$R_1 = (A, B) \quad (\text{using } A \rightarrow B)$$
$$R_2 = (C, B) \quad (\text{using } C \rightarrow B)$$

4. **Check for BCNF:**
   Check if each decomposed relation is in BCNF. A relation is in BCNF if, for every non-trivial functional dependency $X \rightarrow Y$, X is a superkey.
   Both R_1 and R_2 are in BCNF because A is a superkey in R_1 and C is a superkey in R_2.
5. **Check for extraneous attributes:**
   Examine if any attributes in the original relation are extraneous in the decomposed relations. In this case, B appears in both R_1 and R_2. However, B is part of the candidate key in both cases, so it is not extraneous.

Therefore, the BCNF decomposition of the given relation R is:

$$R_1 = (A, B)$$
$$R_2 = (C, B)$$

🔄🔑

No, the given relation does not contain any extraneous attributes.

Q-32: Why do we need triggers in a database? Write the syntax of creating Triggers in SQL. Write the limitation of the Triggers in the database

Answer:

Why We Need Triggers in a Database:

- Automate Data Integrity Enforcement: Triggers proactively execute when specific events occur, ensuring data consistency and integrity without manual intervention.
- Maintain Data Referential Integrity: They enforce relationships between tables, automatically updating or deleting related data to prevent inconsistencies.
- Implement Business Rules: Triggers encapsulate business logic within the database, ensuring consistent application even when data is modified through different channels.
- Audit Data Changes: They can record detailed logs of data modifications for auditing purposes, tracking who made changes, when, and what was changed.
- Create Derived Data: Triggers can automatically populate calculated fields or maintain summary tables, reducing application-side processing.

Syntax for Creating Triggers in SQL:

```
SQL
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
BEGIN
  -- Trigger body (SQL statements to be executed)
END;
```

Limitations of Triggers in Databases:

- Performance Overhead: Triggers can slow down database operations due to additional processing for each affected row.
- Debugging Challenges: Troubleshooting trigger-related issues can be complex, as they often involve multiple actions and layers of logic.
- Cascading Triggers: Triggers can inadvertently trigger other triggers, leading to potential recursion and unexpected behavior if not carefully designed.
- Security Considerations: Triggers can bypass application-level security measures, making it crucial to implement proper authorization within triggers.
- Database Dependency: Triggers are database-specific, making them less portable across different database systems.

Q-33: What is normalization? Why is normalization necessary in database? Explain. Write the advantages and disadvantages of it. [pre-3:1_2021]

Answer:

Normalization is a process used in database design to reduce redundancy and improve data integrity by organizing data into tables and establishing relationships between them. It involves breaking down a database into smaller, more manageable tables and defining rules (normal forms) that govern the relationships between those tables.

Normalization is necessary in databases for several reasons:

1. Reducing data redundancy: By organizing data into separate tables and eliminating duplicate information, normalization helps to minimize data redundancy.

2. Improving data integrity: Normalization helps to maintain data integrity by enforcing logical consistency and eliminating update anomalies. It ensures that data is stored in a structured and consistent manner, reducing the chances of incorrect or inconsistent information being stored.

3. Enhancing query efficiency: Normalization allows for more efficient querying and retrieval of data.

Advantages of normalization:

- Improved data consistency: Normalization minimizes data duplication and inconsistencies, leading to more reliable and consistent data.

- Simplified updates and modifications: Normalized databases experience fewer update anomalies, making it easier to update or modify data without affecting other parts of the database.

- Enhanced data querying and retrieval: Normalization improves query performance by reducing the amount of unnecessary data that needs to be searched, resulting in faster and more efficient query execution.

Disadvantages of normalization:

- Increased complexity: Normalization can result in a more complex database structure with multiple tables and relationships. This complexity can make it harder to understand and maintain the database design.

- Potential performance impact: In some cases, highly normalized databases may experience a slight performance impact due to the need for joining tables to retrieve related data. However, modern database management systems and query optimization techniques help mitigate this impact.

- Over-normalization: Overusing normalization can lead to excessive fragmentation of data across multiple tables, which may make certain operations more complex and time-consuming.

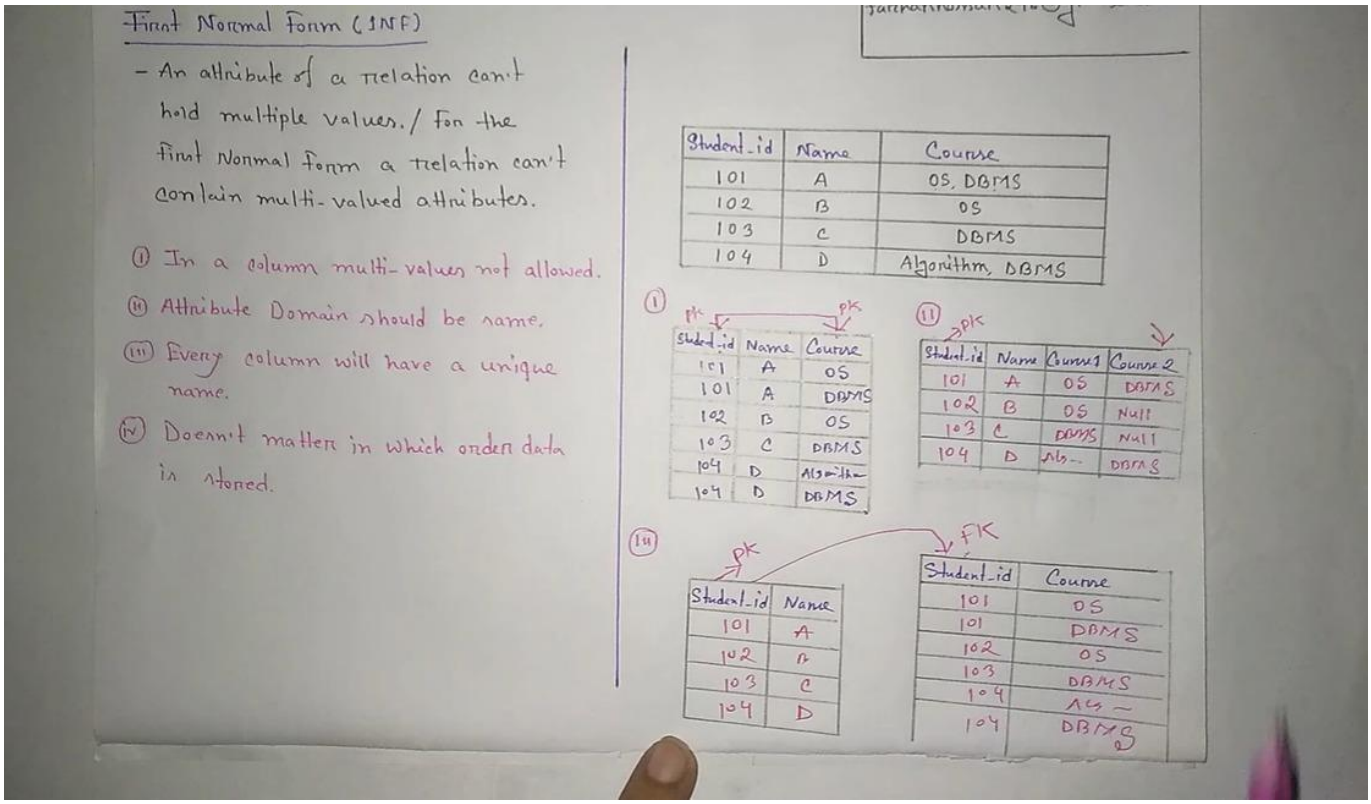
Q-34: Mention the requirements for a good normalized set of tables. How does it differ from 3NF? Explain 1NF, 2NF, 3NF, and BCNF with an example.

Answer:

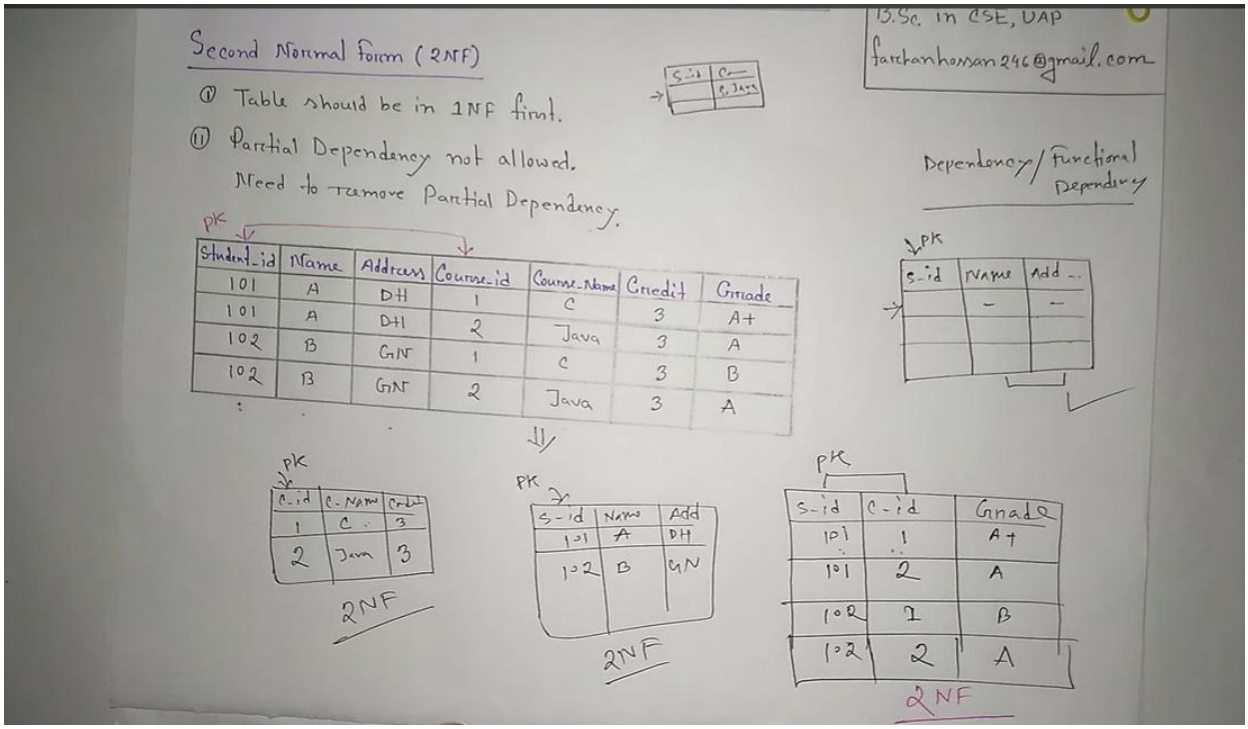
To have a good normalized set of tables, several requirements need to be met. These requirements are based on the principles of database normalization and aim to improve data integrity, eliminate data redundancy, and minimize data anomalies. Here are the key requirements for a well-normalized set of tables:

1. Elimination of Redundancy: Redundant data should be minimized or eliminated. Each piece of data should be stored only once to avoid inconsistencies and save storage space.
2. Data Consistency: The normalization process should strive to maintain data consistency throughout the database. Data dependencies should be properly defined and enforced using primary keys, foreign keys, and referential integrity constraints.
3. Minimization of Update Anomalies: Data should be organized in a way that minimizes the likelihood of update anomalies. This means that modification or deletion of data in one place should not result in inconsistencies or loss of data elsewhere.
4. Preservation of Data Integrity: The database should maintain its integrity by enforcing constraints and relationships between tables. This ensures that data remains accurate, valid, and meaningful.
5. Optimal Query Performance: Normalized tables should allow for efficient and effective querying. Proper indexing and selection of appropriate data types can help optimize query performance.
6. Scalability and Flexibility: A good normalized design should be scalable and adaptable to changing business requirements. It should allow for easy modification, addition, and deletion of data without affecting the overall structure.
7. Ease of Data Modification:The structure should support easy insertion, deletion, and modification of data without introducing anomalies.
8. Optimization of Query Performance: Queries should be able to retrieve the required information efficiently.

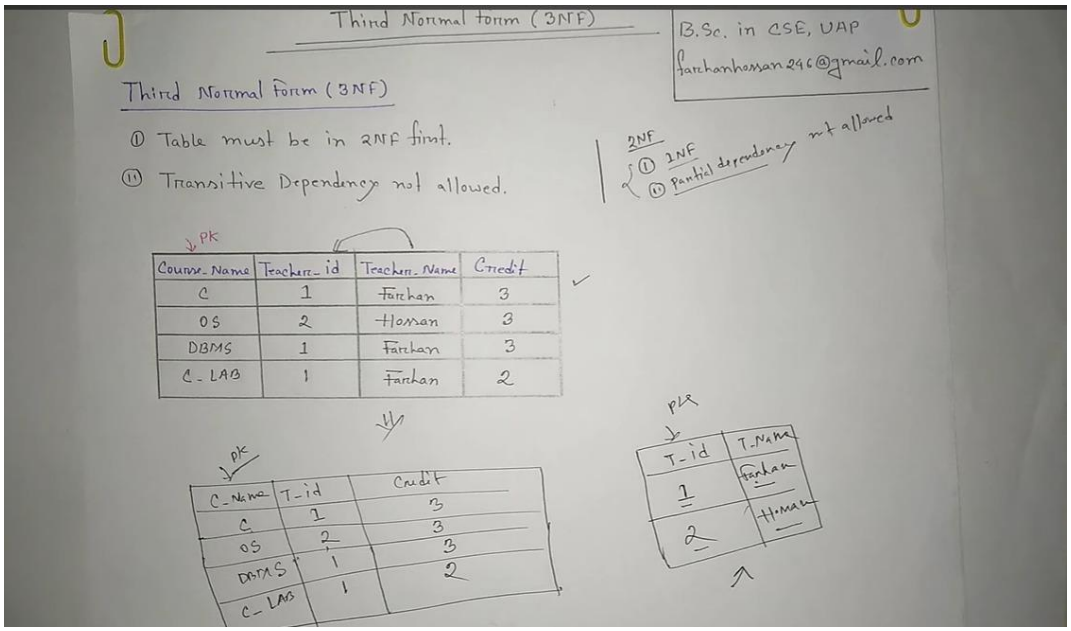
1NF, 2NF, 3NF, and BCNF:



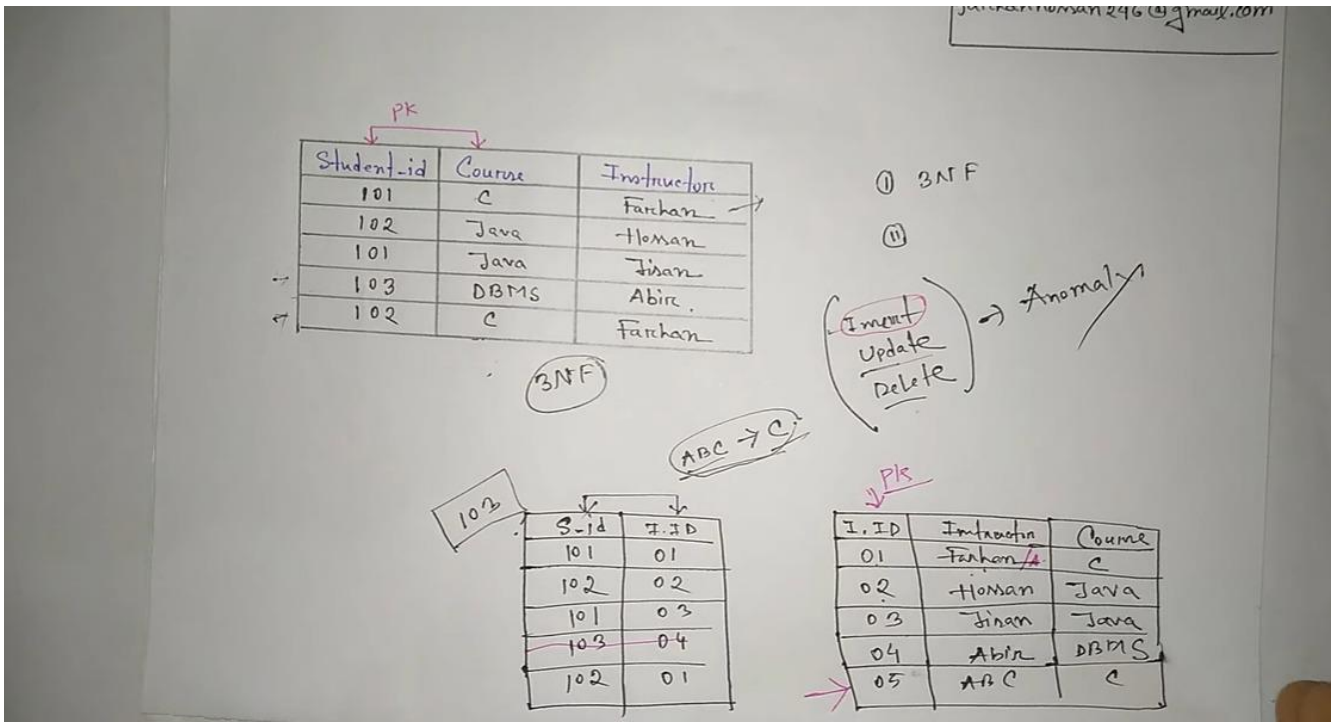
2NF:



3NF:


Third Normal form (3NF)
B.Sc. in CSE, UAP
farhanhossan246@gmail.com
Third Normal form (3NF)
① Table must be in 2NF first.
② Transitive Dependency not allowed.
2NF
① 1NF
② Partial dependency not allowed
PK
Course_Name | Teacher_id | Teacher_Name | Credit
C | 1 | Farhan | 3
OS | 2 | Hosnan | 3
DBMS | 1 | Farhan | 3
C_LAB | 1 | Farhan | 2
PK
C_Name | T_id | Credit
C | 1 | 3
OS | 2 | 3
DBMS | 1 | 2
C_LAB | 1 | 2
PK
T_id | T_Name
1 | Farhan
2 | Hosnan

BCNF:


PK
Student_id | Course | Instructor
101 | C | Farhan
102 | Java | Hosnan
101 | Java | Jisan
103 | DBMS | Abir
102 | C | Farhan
① 3NF
②
Insert
Update
Delete
Anomaly
3NF
ABC → C
103
S_id | T_ID
101 | 01
102 | 02
101 | 03
103 | 04
102 | 01
PK
I_ID | Instructor | Course
01 | Farhan | C
02 | Hosnan | Java
03 | Jisan | Java
04 | Abir | DBMS
05 | ABC | C

Q-35: Discuss Boyce-Codd Normal Form with an example. [pre-3:1_2021]

Answer:

Boyce-Codd Normal Form (BCNF) Explained with an Example

Boyce-Codd Normal Form (BCNF) is a level of database normalization that helps ensure data integrity by eliminating certain types of functional dependencies. It is based on the concept of keys and functional dependencies within a relational database schema.

Why is BCNF important?

- Reduces data redundancy: Eliminates unnecessary duplication of data, minimizing storage requirements and potential inconsistencies.
- Improves data integrity: Enforces stricter data dependencies, minimizing the chance of errors and inconsistent updates.
- Simplifies data manipulation: Queries and updates become more efficient and reliable due to clearer data organization.

Example:

Imagine you have a database table storing information about student enrollments in courses:

Student ID	Course Name	Professor	Grade
101	Math	Mr. Smith	A
101	Physics	Ms. Jones	B
102	Math	Mr. Smith	C
102	English	Ms. Brown	D

This table is in 3NF but not BCNF. Why? Because "Professor" depends on "Course Name," not just "Student ID." Both "Student ID" and "Course Name" together are needed to uniquely identify a professor assignment, but "Course Name" alone isn't enough.

To achieve BCNF, we can split the table into two:

1. Student Enrollment: | Student ID | Course Name |
2. Professor Assignment: | Course Name | Professor |

Now, in both tables, every attribute depends directly on the key:

- In "Student Enrollment," both "Student ID" and "Course Name" together uniquely identify a record.
- In "Professor Assignment," "Course Name" uniquely identifies the professor assigned.

Q-36: Write the differences between "Cartesian product" and "Natural join" operations of relational algebra. [pre-3:1_2021]

Answer:

Aspect	Cartesian Product	Natural Join
Operation	Combines every row from both tables.	Combines rows based on common columns.
Resulting Table Size	Size of Table1 × Size of Table2	Varies based on common column matches.
Common Columns Handling	No consideration of common columns.	Matches and includes only one instance of common columns.
Output Columns	Contains all columns from both tables.	Contains common columns once and other columns from both tables.
Duplicate Columns	Columns with the same name may exist.	Common columns appear only once.
Example	Result includes all possible combinations.	Result includes matched rows based on common columns.

Q-37:Define transaction. Explain the states of transaction. [pre-3:1_2020]

Answer:

😊

A transaction refers to a logical unit of work performed on a database. It involves a sequence of one or more database operations, such as read, write, or modify, which are treated as a single, indivisible unit. Transactions are used to ensure the integrity and consistency of data within a database.

Now, let's talk about the states of a transaction. A transaction goes through different states during its execution. The commonly recognized states are:

1. Active: The initial state of a transaction.

2. Partially committed: When a transaction reaches this state, it has completed all its operations successfully, but the changes it made are not yet permanently saved in the database.

3. Committed: In this state, the changes made by the transaction are permanently saved in the database.

4. Failed: If a transaction encounters an error or fails to complete its operations, it enters the failed state.

5. Aborted: When a transaction is aborted, it means that its changes are rolled back, and the database is restored to its state before the transaction started.

6. Terminated: The final state of a transaction. After a transaction is committed or aborted, it reaches the terminated state, indicating that its execution has completed.

These transaction states help in managing data consistency, ensuring that the database remains in a reliable and stable state even in the event of errors or system failures. 😊🔀

Extra Preparation:

Q-11: What do you mean by an aggregate function? Write down the Aggregate function of a Database query. Explain it with suitable example. [NIT_C_01]

Answer:

An aggregate function is a function in a database query language (such as SQL) that performs a calculation on a set of values and returns a single value as a result.

Common aggregate functions:

- COUNT: Counts the number of rows in a column. Example: SELECT COUNT(*) FROM Customers; counts all customers.
- SUM: Adds up all values in a column. Example: SELECT SUM(TotalSales) FROM Orders; calculates total sales.
- AVG: Calculates the average of values in a column. Example: SELECT AVG(ProductPrice) FROM Products; finds the average product price.
- MIN: Returns the minimum value in a column. Example: SELECT MIN(OrderDate) FROM Orders; finds the earliest order date.
- MAX: Returns the maximum value in a column. Example: SELECT MAX(EmployeeSalary) FROM Employees; finds the highest salary.

Q-12: Explain the difference between data and information.

Answer:

Data	Information
• Data refers to raw facts that have no specific meaning.	• Information refers to processed data that has a purpose and meaning.
• The word 'data' is derived from the Latin word 'datum', which means 'something that is given'.	• The word 'information' is derived from the Latin word 'informatiō', which means 'formation or conception'.
• The data is independent of the information.	• Information is dependent on data.
• Data or raw data is not enough to make a decision.	• The information is sufficient to help make a decision in the respective context.

Q-13: What are the different types of database end users? Discuss the main activities of each.

Answer:

Database users come in all shapes and sizes, each with their own unique needs and interactions with the data.

Database users come in many flavours:

- Casual Flyers: Pre-built tools for simple tasks.
- Data Detectives: Craft queries to analyze and report.
- Code Builders: Design databases and write tools for interaction.
- Business Brains: Translate needs into technical requirements.
- Data Guardians: Keep the database healthy and secure.
- Quick Checkers: Grab specific info and head out.

Q-15: What do you know about TCL? Discuss two TCL commands used in SQL with examples. [Pre_2021]

Answer:

TCL (Transaction Control Language) is a subset of SQL commands that manage database transactions, ensuring data consistency and integrity.

1. COMMIT: The COMMIT command is used to permanently save the changes made in a transaction. It marks the successful completion of the transaction, making all the changes made within the transaction become permanent and visible to other users.

Example:

```
BEGIN TRANSACTION;
-- Perform some database operations (INSERT, UPDATE, DELETE, etc.)
COMMIT;
```

2. ROLLBACK: The ROLLBACK command is used to undo or cancel the changes made within a transaction. It rolls back the database to the previous consistent state before the transaction was initiated.

Example:

```
BEGIN TRANSACTION;
-- Perform some database operations (INSERT, UPDATE, DELETE, etc.)
IF <some condition> THEN
    ROLLBACK;
ELSE
    COMMIT;
END IF;
``
```

Q-16: Description of DBMS Features to manage data security.

Answer:

- Authentication & Authorization: Verify users and control access based on roles and permissions.
- Encryption: Protect data at rest and in transit with strong encryption algorithms.
- Auditing & Logging: Track user activity and database changes for accountability and security analysis.
- Access Control Lists: Define precise data access permissions for granular control.
- Data Masking & Anonymization: Securely obscure sensitive data for authorized use while protecting privacy.
- Firewalls & IDS: Filter traffic and monitor activity for potential threats.
- Secure Backups & Recovery: Ensure data availability in case of disruptions.
- Vulnerability Management: Regularly scan and patch vulnerabilities to minimize attack risks. 9. Data Breach Response: Respond swiftly and responsibly to any security incidents.

Relational Model and Algebra:

Q-1: How can you represent the average balance of customers using relational algebra? [Use the Figure 1] [pre-3:1_2021]

Answer:

To represent the average balance of customers using relational algebra, you can use the following expression:

π customer-name, AVG(balance)(σ account.branch-name = branch.branch-name ∧ branch.branch-city = 'City')(account ⋈ branch)

Here, π represents the projection operation, AVG represents the average aggregate function, σ represents the selection operation, ⋈ represents the natural join operation, and 'City' represents the desired city for calculation.

Q-02: [pre-3:1_2021]

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)

Give an expression in the relational algebra for each of the following queries:
i) Find all loan numbers with a loan value greater than $10,000.
ii) Find the names of all depositors who have an account with a value greater than $6,000.
iii) Find the names of all borrowers who have a loan in branch "Downtown".

Answer:

i) To find all loan numbers with a loan value greater than $10,000, you can use the following expression:

π loan-number (σ amount > 10000)(loan)

ii) To find the names of all depositors who have an account with a value greater than $6,000, you can use the following expression:

π customer-name (σ balance > 6000)(depositor ⋈ account)

iii) To find the names of all borrowers who have a loan in branch "Downtown", you can use the following expression:

π customer-name (σ branch-name = 'Downtown')(borrower ⋈ loan)

E-R Model:

Q-1: What is ER modelling?

Answer:

ER modeling is a conceptual modeling technique used to represent and analyze the structure of a database system. It helps to visualize and understand the relationships between entities (objects or concepts) within a domain.

Q-3: What do you understand by generalization and specialization? Explain.

Answer:

Generalization: Generalization is the process of extracting common properties or characteristics from two or more entities and creating a more general entity to represent them. It focuses on identifying shared attributes and behaviors among entities to form a superclass or higher-level entity.

Specialization is the opposite process of generalization. It involves creating new entities (subclasses) from an existing entity (superclass) by adding specific attributes or behaviors that are unique to those entities. Specialization allows for representing entities that have distinct characteristics or requirements.

Q-4: Define relationship and relationship set. Explain the different mapping cardinality of a binary 1 relationship set.

Answer:

Relationship:

- **Relationship** refers to an association or connection between two or more entities within a database. It represents how entities are related to each other.

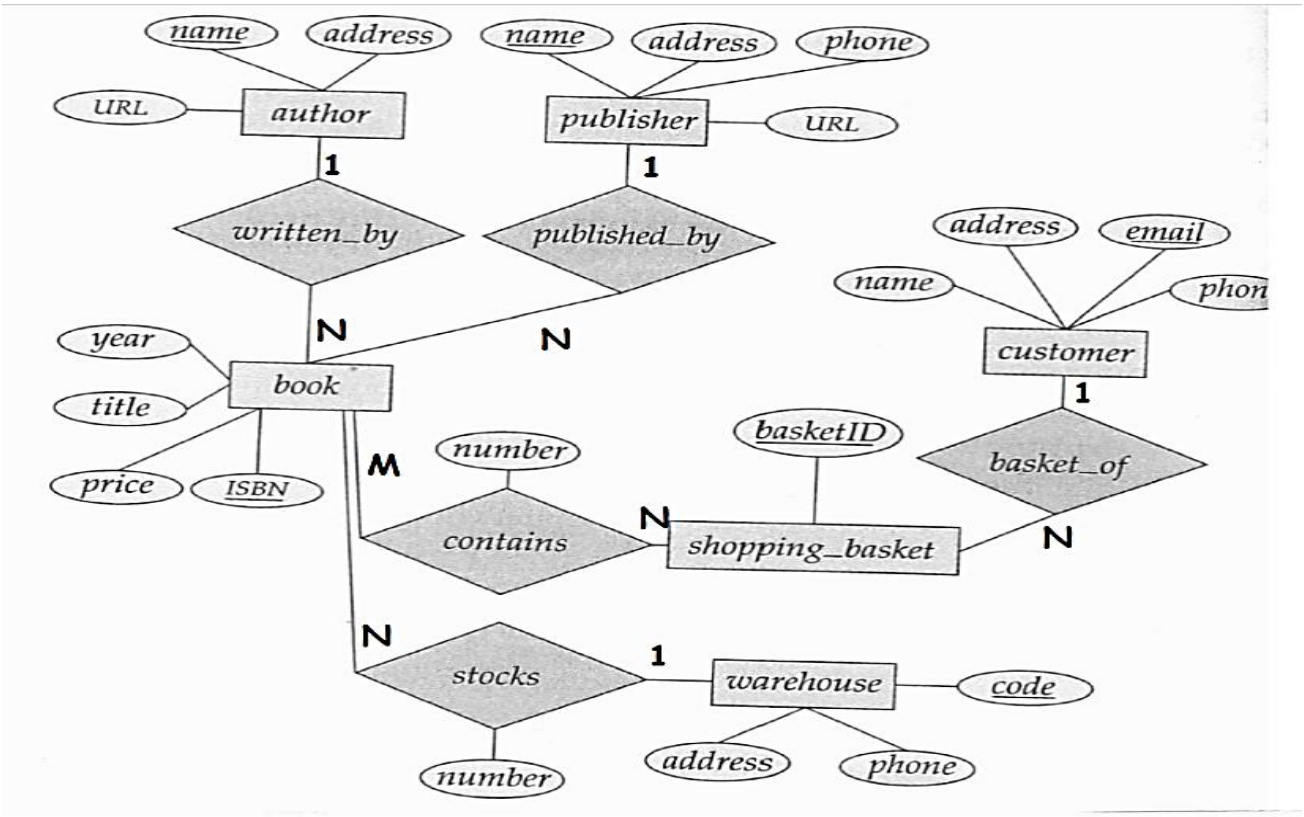
Relationship Set:

- **Relationship Set** is a collection of similar relationships that share the same attributes or characteristics.

Q-5: Construct the E-R diagram for the following relational schemas: [pre-3:1_2020]

- Books (**ISBN**, title, price, category)
- Publishers (**publisher id**, name, year of publications)
- Readers (**user id**, name, email, phone no, address)
- Staff's (**staff id**, name)

Answer:



E-R DIAGRAM OF ONLINE BOOKSTORE

SOL:

Q-01: What is view? Why it needed? [MEC_C_01] [STEC_C_01]

Answer:

Views are virtual tables derived from existing tables in a database. They act as customized lenses, filtering and presenting specific data to users according to their needs and permissions.

Benefits of Views:

- Simplified querying: Users can access and manipulate data through a simplified view instead of complex, underlying queries.
- Data security: Views can restrict access to sensitive data by only exposing relevant columns or rows.
- Data consistency: Changes made to the underlying table automatically propagate to the view, ensuring data integrity.
- Logical data organization: Views can present data in a way that aligns with specific user roles or tasks, improving clarity and comprehension.
- Reduced redundancy: Duplicate data can be eliminated by using views instead of creating and maintaining multiple copies.

Q-02: Write the SQL queries and output from that queries of Right outer join and Full outer join using following relations. [MEC_C_01]

CLIENTS

ID	NAME_	AGE	CITY	WAGE
1	Rex	32	Ahmedabad	2000.00
2	Komal	25	Delhi	1500.00
3	Kurt	23	Kota	2000.00
4	Cassie	25	Mumbai	6500.00
5	Herald	27	Bhopal	8500.00
6	Kim	22	MP	4500.00
7	Maddy	24	Indore	1000.00

ORDERS

OID	DATE	CLIENT_ID	BILL
100	2019-10-18	3	1500
101	2019-11-21	2	1560
102	2019-10-18	3	3000
103	2018-05-21	4	2060

Answer:

Right Outer Join:

SQL query:

```
SELECT CLIENTS.ID, CLIENTS.NAME_, CLIENTS.AGE, CLIENTS.CITY,
CLIENTS.WAGE, ORDERS.OID, ORDERS.DATE, ORDERS.BILL

FROM CLIENTS

RIGHT JOIN ORDERS

ON CLIENTS.ID = ORDERS.CLIENT_ID;
```

Output:

```
Plain Text

ID  NAME_   AGE CITY    WAGE    OID DATE    BILL
3   Kurt    23  Kota    2000.00 100 2019-10-18  1500
3   Kurt    23  Kota    2000.00 102 2019-10-18  3000
2   Komal   25  Delhi   1500.00 101 2019-11-21  1560
4   Cassie  25  Mumbai  6500.00 103 2018-05-21  2060
```

Full Outer Join:

SQL query:

```
SELECT CLIENTS.ID, CLIENTS.NAME_, CLIENTS.AGE, CLIENTS.CITY,
CLIENTS.WAGE, ORDERS.OID, ORDERS.DATE, ORDERS.BILL

FROM CLIENTS

FULL OUTER JOIN ORDERS

ON CLIENTS.ID = ORDERS.CLIENT_ID;
```

Output:

```
Plain Text

ID  NAME_   AGE CITY    WAGE    OID DATE    BILL
1   Rex 32  Ahmedabad   2000.00 NULL    NULL    NULL
2   Komal   25  Delhi   1500.00 101 2019-11-21  1560
3   Kurt    23  Kota    2000.00 100 2019-10-18  1500
3   Kurt    23  Kota    2000.00 102 2019-10-18  3000
4   Cassie  25  Mumbai  6500.00 103 2018-05-21  2060
5   Herald  27  Bhopal  8500.00 NULL    NULL    NULL
6   Kim 22  MP  4500.00 NULL    NULL    NULL
7   Maddy   24  Indore  1000.00 NULL    NULL    NULL
```

Note: In the Full Outer Join output, NULL values indicate that there is no matching record in the respective table for the join condition.

Q-03: Write the general syntax of the CREATE TABLE statement in SQL. Write the code for creating a table of the schema: book (ISBN, title, year, price). [pre-3:1_2021]

Answer:

The general syntax of the CREATE TABLE statement in SQL is as follows:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ...
);
```

To create a table of the schema: book (ISBN, title, year, price), you can use the following code snippet:

```
CREATE TABLE book (
    ISBN VARCHAR(20),
    title VARCHAR(100),
    year INT,
    price DECIMAL(10, 2)
);
```

Q-04: Write the roles of each clause used in an SQL query. What is the role of each clause? Write the typical form of an SQL query. [pre-3:1_2021]

Answer:

In an SQL query, each clause has a specific role and contributes to defining the purpose and behavior of the query. Here are the roles of some common clauses used in an SQL query:

1. SELECT: Specifies the columns or expressions to be retrieved from the database. It is used to define the projection of the query.
2. FROM: Specifies the table(s) from which the data is retrieved. It defines the source of the data for the query.
3. WHERE: Filters the data based on specific conditions. It is used to define the selection criteria for the query.
4. GROUP BY: Groups the rows based on one or more columns. It is used to aggregate data and perform calculations.

5. HAVING: Filters the grouped data based on specific conditions. It is similar to the WHERE clause but is applied to the grouped data.
6. ORDER BY: Sorts the result set based on one or more columns. It is used to specify the order in which the result set should be presented.
7. JOIN: Combines rows from two or more tables based on a related column between them. It is used to retrieve data from multiple tables.

The typical form of an SQL query is as follows:

```
SELECT column1, column2, ...
FROM table1
JOIN table2 ON condition
WHERE condition
GROUP BY column(s)
HAVING condition
ORDER BY column(s);
```

Q-05: Consider the database schema below: [pre-3:1_2021]

employee (person-name, street, city)
works (person-name, company name, salary)
company (company-name, city)
manages (person-name, manager name)

Provide SQL expressions for the following queries:

i. Find the names of all employees who live in the same city and on the same street as their managers.
ii. Find the names of all employees in this database who do not work for "First Bank Corporation".
iii. Find the names of all employees who earn more than every employee of "Small Bank Corporation".
iv. Find the company with the most employees.

Answer:

To answer the queries based on the given database schema, you can use SQL expressions as follows:

i. Find the names of all employees who live in the same city and on the same street as their managers.

```
SELECT e.person_name
FROM employee e
JOIN manages m ON e.city = m.city AND e.street = m.street
```

ii. Find the names of all employees in this database who do not work for "First Bank Corporation".

```
SELECT e.person_name
FROM employee e
WHERE e.person_name NOT IN (
    SELECT w.person_name
    FROM works w
    WHERE w.company_name = 'First Bank Corporation'
)
```

iii. Find the names of all employees who earn more than every employee of "Small Bank Corporation.

```
SELECT e.person_name
FROM employee e
WHERE e.salary > ALL (
    SELECT w.salary
    FROM works w
    WHERE w.company_name = 'Small Bank Corporation'
)
```

iv. Find the company with the most employees.

```
SELECT c.company_name
FROM company c
JOIN works w ON c.company_name = w.company_name
GROUP BY c.company_name
ORDER BY COUNT(*) DESC
LIMIT 1
```

Note: The queries assume that the column names in the respective tables are as mentioned in the database schema. Adjust the expressions accordingly if the actual column names differ.

Q-06: Consider the following relational database: [pre-3:1_2020]

Employees: Employee name, street, city

Works: Employee name, company name, salary

Write a SQL query to find the followings:

I. Find the name of all employees whose salary is greater than BDT 100000.

II. Find the name of all employee who lives in city "DHAKA".

III. Find the name of all employees who lives in city "DHAKA" and salary is greater than BDT 100000.

Answer:

To find the required information from the given relational database, you can use the following SQL queries:

I. Find the name of all employees whose salary is greater than BDT 100000.

```
SELECT Employee_name
FROM Works
WHERE Salary > 100000;
```

II. Find the name of all employees who live in city "DHAKA".

```
SELECT Employee_name
FROM Employees
WHERE city = 'DHAKA';
```

III. Find the name of all employees who live in city "DHAKA" and salary is greater than BDT 100000.

```
SELECT E.Employee_name
FROM Employees E
INNER JOIN Works W ON E.Employee_name = W.Employee_name
WHERE E.city = 'DHAKA' AND W.Salary > 100000;
```

Note: Ensure that the actual column names in your database match the ones mentioned in the queries.

Q-07: The general syntax for typical SQL Data Manipulation Language (DML) statements is as follows:

1. INSERT: Used to insert new records into a table.

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

2. SELECT: Used to retrieve data from one or more tables.

```
SELECT column1, column2, ...
FROM table_name;
```

3. UPDATE: Used to modify existing records in a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

4. DELETE: Used to delete records from a table.

```
DELETE FROM table_name
WHERE condition;
```

Regarding the comparison between `<> 'ALL'` and `NOT IN`, they are not identical in SQL. The `<>` operator is used to check for inequality between two values, while `NOT IN` is used to check if a value is not present in a specified set.

For example, consider the following query:

```
SELECT column1
FROM table_name
WHERE column1 <> 'ALL';
```

This query will retrieve all the records from `table_name` where the value in `column1` is not equal to 'ALL'.

On the other hand, the `NOT IN` operator is used to check if a value is not present in a specified set. Here's an example:

```
SELECT column1
FROM table_name
WHERE column1 NOT IN ('Value1', 'Value2', 'Value3');
```

This query will retrieve all the records from `table_name` where the value in `column1` is not equal to any of the specified values ('Value1', 'Value2', 'Value3').

Q-08: How does pattern match of strings work in SQL? Give an appropriate example of the pattern matching.

Answer:

Pattern matching in SQL is achieved using the LIKE operator with wildcard characters. The wildcard characters used in pattern matching are:

- `%` (percent sign): Matches any sequence of zero or more characters.

- `_` (underscore): Matches any single character.

Here's an example of pattern matching in SQL:

Consider a table called `Employees` with columns `EmployeeID` and `EmployeeName`. We want to retrieve all employees whose names start with 'J' and end with 'son'. We can use the LIKE operator with wildcard characters to achieve this:

```
SELECT EmployeeName

FROM Employees

WHERE EmployeeName LIKE 'J%son';
```

This query will return all the employees whose names start with 'J' and end with 'son', such as 'Jackson', 'Johnson', 'Jameson', etc. The `%` wildcard matches any sequence of characters between 'J' and 'son'.

Q-09: Consider the database schema below: [pre-2:2_2021]

- *member (memb_no, name, age)*
- *book (ISBN, title, authors, publisher)*
- *borrowed (memb_no, ISBN, date)*

Figure: Library database

Give an expression in SQL for the following queries:

i. Print the names of members who have borrowed any book published by "Springer".
ii. Print the names of members who have borrowed all books published by "MDPI".
iii. For each publisher, print the names of members who have borrowed more than five books of that publisher.
iv. Print the average number of books borrowed per member. Take into account that if a member does not borrow any books, then that member does not appear in the borrowed relation at all.

Answer:

i. To print the names of members who have borrowed any book published by "Springer", you can use a combination of the `member`, `book`, and `borrowed` tables along with the `JOIN` and `LIKE` operators. Here's the SQL expression:

```
SELECT m.name
FROM member m
JOIN borrowed b ON m.memb_no = b.memb_no
JOIN book bo ON b.ISBN = bo.ISBN
WHERE bo.publisher LIKE '%Springer%';
```

ii. To print the names of members who have borrowed all books published by "MDPI", you can use a combination of the `member`, `book`, and `borrowed` tables, along with the `JOIN` and `GROUP BY` clauses. Here's the SQL expression:

```
SELECT m.name
FROM member m
JOIN borrowed b ON m. memb_no = b. memb_no
JOIN book bo ON b. ISBN = bo. ISBN
WHERE Bo. Publisher = 'MDPI'
GROUP BY m.name
HAVING COUNT (DISTINCT bo. ISBN) = (SELECT COUNT (DISTINCT ISBN) FROM book WHERE publisher = 'MDPI');
```

iii. To print the names of members who have borrowed more than five books from each publisher, you can use a combination of the `member`, `book`, and `borrowed` tables, along with the `JOIN`, `GROUP BY`, and `HAVING` clauses. Here's the SQL expression:

```
SELECT p.publisher, m.name
FROM member m
JOIN borrowed b ON m.memb_no = b.memb_no
JOIN book bo ON b.ISBN = bo.ISBN
JOIN (
   SELECT publisher, COUNT(DISTINCT ISBN) AS book_count
   FROM book
   GROUP BY publisher
```

```
) AS p ON bo.publisher = p.publisher
WHERE p.book_count > 5
GROUP BY p.publisher, m.name;
```

iv. To print the average number of books borrowed per member, taking into account that members who have not

```
SELECT AVG(num_borrowed) AS avg_borrowed_per_member
FROM (
    SELECT m.memb_no, COUNT(b.ISBN) AS num_borrowed
    FROM member m
    LEFT JOIN borrowed b ON m.memb_no = b.memb_no
    GROUP BY m.memb_no
) AS borrowed_counts;
```

Q-09: Consider the following two schemas: [pre-2:2_2021]

salaries (emp_num, valid from, amount)

salary archives (emp_num, valid from, amount)

Now create a BEFORE DELETE trigger that inserts a new record into the salary archives table before a row is deleted from the salaries table.

Answer:

Here's an example of a BEFORE DELETE trigger that inserts a new record into the salary archives table before a row is deleted from the salaries table:

```
CREATE TRIGGER before_delete_trigger
BEFORE DELETE ON salaries
FOR EACH ROW
BEGIN
    -- Insert the deleted row into salary archives table
    INSERT INTO salary_archives (emp_num, valid_from, amount)
    VALUES(OLD.emp_num, OLD.valid_from, OLD.amount);
END;
```

In this trigger, the **BEFORE DELETE** clause ensures that the trigger is executed before a row is deleted from the **salaries** table. The **FOR EACH ROW** clause specifies that the trigger is fired for each affected row.

The trigger logic then uses the **INSERT INTO** statement to insert the values from the deleted row (**OLD.emp_num**, **OLD.valid_from**, and **OLD.amount**) into the **salary_archives** table.

By using this trigger, whenever a row is deleted from the **salaries** table, a copy of that row will be inserted into the **salary_archives** table before the deletion takes place.

Q-10: Write the syntax of creating a view in SQL. Create a view to show the name of the gardeners who use fertilizer for their plant shown in Figure 3. [pre-2:2_2021]

Answer:

To create a view in SQL, you can use the **CREATE VIEW** statement followed by the name of the view and the columns it will include. Here's the syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

To create a view that shows the name of the gardeners who use fertilizer for their plants, you can use the following SQL query:

```
CREATE VIEW gardener_fertilizer_view AS
SELECT g.first_name, g.last_name
FROM gardener g
JOIN planting p ON g.gid = p.gid
WHERE p.fertilizer = 'yes';
```

This view joins the **gardener** table with the **planting** table on the **gid** column and selects the **first_name** and **last_name** columns from the **gardener** table. It also includes a condition to only include rows where the **fertilizer** column in the **planting** table is set to 'yes'. The resulting view will display the names of gardeners who use fertilizer for their plants.

Q-11: What do you know about left and right outer join? Give an example of each join using the following two relations in Figure and show the corresponding outputs. [pre-2:2_2021]

gid	*first name*	*last name*	*birthday*
1	*Albert*	*Einstein*	*1879-03-14*
2	*Albert*	*Slater*	*1973-10-10*
3	*Christian*	*Slater*	*1969-08-18*
4	*Christian*	*Gardener*	*1974-01-30*

Gardener

pid	*gid*	*plant_name*	*fertilizer*	*planting_date*
1	*3*	*rose*	*yes*	*2001-01-15*
2	*4*	*daisy*	*yes*	*2020-05-16*
3	*8*	*rose*	*no*	*2005-08-10*
6	*1*	*sunflower*	*yes*	*2015-08-20*
7	*6*	*violet*	*yes*	*1997-01-17*

Planting

Figure: Join table

Answer:

The syntax for a left outer join is as follows:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

In a left outer join, all the rows from the left table (table1) are included, along with matching rows from the right table (table2). If there are unmatched rows in the right table, NULL values are used for the columns of the right table in the output.

Here's an example using the given tables:

```
SELECT g.gid, g.first_name, g.last_name, p.plant_name
FROM gardener g
```

```
LEFT JOIN planting p
ON g.gid = p.gid;
```

This will return all rows from the gardener table and the matching rows from the planting table based on the gid column. Unmatched rows in the planting table will have NULL values for the columns of the planting table in the output.

The syntax for a right outer join is similar:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

In a right outer join, all the rows from the right table (table2) are included, along with matching rows from the left table (table1). If there are unmatched rows in the left table, NULL values are used for the columns of the left table in the output.

Example using the given tables:

```
SELECT p.gid, g.first_name, g.last_name, p.plant_name
FROM gardener g
RIGHT JOIN planting p
ON g.gid = p.gid;
```

This will return all rows from the **planting** table and the matching rows from the **gardener** table based on the **gid** column. Unmatched rows in the **gardener** table will have NULL values for the columns of the **gardener** table in the output.

Please note that the outputs will vary based on the actual data present in the tables.

- SQL clauses with logical operators.
- SQL functions with syntax.
- SQL JOINS with diagram.
- SQL statements to create a Primary Key.

Normalization:

Q-04: Suppose the table and functional dependency are as follows: [pre-3:1_2020]

SID	Major	Advisor
123	Physics	Faculty - 1
123	Music	Faculty - 2
456	Bio	Faculty - 3
789	Physics	Faculty – 4
999	Physics	Faculty - 1

Functional dependencies in the relation:

- SID, Major → Advisor
- Advisor → Major

Answer:

Q-05: Find out the Boyce-Codd Normal Form (BCNF) of the above table. [pre-3:1_2020]

Answer:

Q-07: Describe the types of update anomalies that may occur on a table that has redundant data. [pre-2:2_2021]

Answer:

When a table contains redundant data, it means that the same information is stored in multiple places within the table. Redundancy can lead to several types of update anomalies, which are situations where inconsistencies or errors arise when performing insert, update, or delete operations on the table. Here are three common types of update anomalies associated with redundant data:

1. Insertion Anomaly: This anomaly occurs when it is not possible to insert certain data into the table without including other unrelated data. In other words, the table's structure and dependencies force users to enter unnecessary or incomplete information. For example, imagine a database where customer information is duplicated in both the sales and billing tables. If a new customer wants to make a purchase but does not require billing services, an insertion anomaly may occur because the system requires them to enter data into the billing table, even though it's not necessary for their transaction.

2. Update Anomaly: An update anomaly arises when modifications to the table's data result in inconsistencies or contradictions. In the presence of redundant data, updating one occurrence of a value may lead to inconsistencies with other occurrences. For instance, consider a product inventory table where product names are redundantly stored along with each individual order. If a product name changes, an update anomaly may occur because the change needs to be made in multiple places throughout the table, and the possibility of missing or inconsistent updates is high.

3. Deletion Anomaly: Deletion anomalies happen when removing data from the table unintentionally removes other necessary related information. Redundant data makes it difficult to selectively delete specific records without losing relevant data from other records. For instance, suppose a table contains customer details along with their orders. If a customer cancels an order, but the table does not store customer details separately, deleting the canceled order would result in the loss of customer information, even though it may still be needed for other active orders.

These anomalies can lead to inconsistent and inaccurate data, which can negatively impact data integrity and query results. To avoid these issues, it's crucial to eliminate redundant data by normalizing the table's structure and removing unnecessary duplicate information. 🔄 🚫

Q-08: Describe the characteristics of a table that violates the first normal form (1NF) and then describe how such a table is converted to 1NF. [pre-2:2_2021]

Answer:

A table that violates the first normal form (1NF) exhibits certain characteristics that indicate data redundancy and incomplete dependencies. Here are the characteristics of a table that violates 1NF:

1. Non-atomic values: The table contains columns with multiple values or sets of values, violating the atomicity principle. For example, a table may have a "Skills" column that stores multiple skills in a single field, such as "Java, Python, SQL" in a single cell.

2. Repeating groups: The table has repeating groups of columns, meaning that similar sets of attributes are stored consecutively. For instance, in a table representing employees, there may be columns like "Phone 1," "Phone 2," "Phone 3," etc., indicating a repeating group of phone numbers.

3. Incomplete dependencies: There are dependencies between attributes that are not fully captured by the table's structure. This happens when an attribute depends on only part of the primary key rather than the entire key. In such cases, changing the value of a partial key can lead to inconsistencies. For example, if a table has a primary key consisting of "OrderID" and "ProductID" but stores "ProductName" based only on the "ProductID," it violates 1NF as the product name depends on only part of the primary key.

To convert a table violating 1NF into first normal form, the following steps can be taken:

1. Identify the primary key: Determine the primary key that uniquely identifies each record in the table. It may consist of one or more attributes.

2. Eliminate repeating groups: Move the repeating groups of columns into separate tables. Each table should have its own primary key and relate to the original table through foreign key relationships.

3. Ensure atomicity: Split any columns with multiple values into separate columns, each storing a single value. This ensures that each attribute has an atomic value. For example, the "Skills" column with "Java, Python, SQL" would be split into individual columns like "Skill1," "Skill2," and "Skill3."

4. Remove partial dependencies: Analyze the dependencies between attributes. If any attributes depend on only a portion of the primary key, create separate tables for those attributes to establish complete dependencies. These new tables will have foreign keys referencing the primary key of the original table.

By following these steps, the table can be transformed into first normal form (1NF) by eliminating data redundancy, ensuring atomicity, and capturing all dependencies accurately.
🔄🧩

Q-09: Given F ={A->C, AB->C), prove that B is extraneous in AB->C. [pre-2:2_2021]

Answer:

To prove that B is extraneous in the functional dependency AB->C, we need to demonstrate that removing B from the left-hand side (LHS) of the functional dependency does not alter the closure of the original set of functional dependencies. In other words, removing B from AB->C should still yield the same closure as before.

Let's start by finding the closure (F+) of the original set of functional dependencies F, which contains AB->C.

Step 1: Start with the original set of functional dependencies F: {A->C, AB->C}

Step 2: Compute the closure of F, denoted as F+, which is the set of all functional dependencies that can be derived from F.

For A->C:

A+ = A (by reflexivity)

Therefore, A+ = {A, C}

For AB->C:

AB+ = AB (by reflexivity)

AB+ = {A, B, C}

Step 3: Now, let's remove B from AB->C to check if it still holds the same closure as before.

New functional dependency: A->C

Step 4: Compute the closure of the new functional dependency set, {A->C}, denoted as NewF+.

For A->C:

A+ = A (by reflexivity)

Therefore, A+ = {A, C}

Comparing the results of Step 2 (F+) and Step 4 (NewF+), we can see that the closure of the new set of functional dependencies without B (A->C) is the same as the closure of the original set of functional dependencies (AB->C).

Since removing B from AB->C still maintains the same closure, B is indeed extraneous in the functional dependency AB->C. 🔍 🔢

Q-10: Define closure of a set of attributes with an algorithm. [pre-2:2_2021]

Answer:

The closure of a set of attributes, denoted as X+, is the set of all attributes that can be determined or derived from the given set of attributes X using a set of functional dependencies. Here's an algorithm to compute the closure of a set of attributes:

1. Start with the given set of attributes X.

2. Set the closure X+ as X initially.

3. For each functional dependency F in the set of functional dependencies F+:

   a. Check if the left-hand side (LHS) of F is a subset of X+. If it is, proceed to the next step. Otherwise, skip this functional dependency.

   b. If the LHS of F is a subset of X+, add the right-hand side (RHS) of F to X+. If any new attributes are added to X+, go back to step 3. This is because the newly added attributes may be the LHS of other functional dependencies, which would allow additional attributes to be derived.

4. Repeat step 3 until there are no more changes to X+.

5. Once there are no more changes to X+, X+ represents the closure of the initial set of attributes X.

Here's a pseudo code representation of the algorithm:

```
function computeClosure(X, F):
    X+ = X
    repeat:
        changes = false
        for each functional dependency F in F:
            if LHS(F) is subset of X+:
                if RHS(F) is not subset of X+:
                    X+ = X+ union RHS(F)
```

```
            changes = true
        until changes = false
    return X+
```

This algorithm iteratively checks and adds attributes to the closure until no more changes are made, ensuring that all derivable attributes from the given set are included in the closure. 💉 🔢

Q-11: Given relation schema R = {A, B, C, D, E} and FDs A->BC, CD->E, B->D, E->A, now find out: [pre-2:2_2021]

i. (A)+, (AB)+, (BC)+, (ABC)+
ii. List the candidate keys for R.

Answer:

To answer your question, let's analyze the given relation schema R and its functional dependencies (FDs):

Relation schema R: {A, B, C, D, E}

Functional dependencies:

1. A->BC

2. CD->E

3. B->D

4. E->A

i. Computing the closures:

1. (A)+

To find (A)+, we start with the attribute A and check which attributes can be derived from it.

Step 1: (A)+ = A

Step 2: Check FDs

- (A)+ = A, AB (adding B because of FD: B->D)

- (A)+ = A, AB, BD (adding D because of FD: CD->E)

- (A)+ = A, AB, BD, CDE (adding C and E because of FD: A->BC)

Therefore, (A)+ = A, AB, BD, CDE.

2. (AB)+

To find (AB)+, we start with the attributes AB and check which attributes can be derived from them.

Step 1: (AB)+ = AB

Step 2: Check FDs

- (AB)+ remains the same as no FD involves both A and B on the left-hand side.

Therefore, (AB)+ = AB.

3. (BC)+

To find (BC)+, we start with the attributes BC and check which attributes can be derived from them.

Step 1: (BC)+ = BC

Step 2: Check FDs

- (BC)+ remains the same as no FD involves both B and C on the left-hand side.

Therefore, (BC)+ = BC.

4. (ABC)+

To find (ABC)+, we start with the attributes ABC and check which attributes can be derived from them.

Step 1: (ABC)+ = ABC

Step 2: Check FDs

- (ABC)+ remains the same as no FD involves all attributes A, B, and C on the left-hand side.

Therefore, (ABC)+ = ABC.

ii. Finding the candidate keys for R:

To determine the candidate keys, we need to find the attribute combinations whose closures include all attributes in R.

From the given FDs, we can make the following observations:

- A candidate key must include attribute A, as A is not derivable from any other attribute.

- Since B->D (FD: B->D) and D is not derivable from any other attribute, B must be included in any candidate key.

- Since E->A (FD: E->A) and A is not derivable from any other attribute, E must also be included in any candidate key.

Possible candidate keys for R are:

- AB

- AE

Q-12: Define deadlock with a suitable example. Discuss different deadlock-prevention schemes using timestamps. [pre-3:1_2021]

Answer:

Deadlock is a situation in which two or more processes are unable to proceed because each is waiting for a resource held by the other. It occurs when there is a circular chain of dependencies among processes, where each process is waiting for a resource that is held by another process in the chain.

Let's consider an example to understand deadlock:

Suppose we have two processes, P1 and P2, and two resources, R1 and R2. The processes require both resources to complete their execution. The following scenario illustrates a deadlock:

1. P1 acquires R1.

2. P2 acquires R2.

3. P1 requests R2 but is blocked because P2 is holding it.

4. P2 requests R1 but is blocked because P1 is holding it.

In this scenario, both processes are waiting for a resource that the other process is holding, resulting in a deadlock. Neither process can continue execution until it acquires the resource held by the other process, leading to a complete halt.

To prevent deadlocks, various schemes can be used, such as timestamp-based approaches. Timestamps are logical values assigned to events or transactions to provide a total order of occurrence.

Here are some deadlock-prevention schemes that utilize timestamps:

1. Wait-Die Scheme:

In this scheme, each process is assigned a timestamp when it enters the system. When a process requests a resource, if the requesting process's timestamp is less than the timestamp of the process holding the resource, the requesting process is allowed to wait. If the timestamps are reversed, the requesting process is rolled back to its initial state (dies) and can be restarted later.

2. Wound-Wait Scheme:

In this scheme, processes are assigned timestamps as well. When a process requests a resource, if the resource is held by another process with a higher timestamp, the requesting process must wait. If the timestamps are reversed, the process holding the resource is rolled back (wounded) to its initial state, and the requesting process is allowed to proceed.

3. Priority-based Scheme:

In this scheme, each process is assigned a priority. When a process requests a resource, the system checks the priority of both the requesting process and the process holding the resource. If

the holding process has a higher priority, the requesting process is made to wait. If the priorities are reversed, the higher-priority process can preempt the resource from the lower-priority process.

These timestamp-based deadlock-prevention schemes help ensure that processes operate in a coordinated and non-blocking manner, reducing the chances of deadlocks. By carefully managing timestamps and resource allocation, deadlocks can be prevented or resolved in an efficient manner. ⏳ 🔒 🚫

Extra:

Q-13:What are the purposes of database recovery? Classify and explain the various types of system failure. [pre-3:1_2021]
Answer:

The purposes of database recovery are:

1. Restore the Database: After a system failure, the database needs to be restored to its last consistent state before the failure occurred.
2. Maintain Data Integrity: Recovery mechanisms ensure that the database remains consistent and reliable, preserving the integrity of the data.
3. Preserve Data Durability: Recovery processes aim to make committed changes permanent and durable by storing them in non-volatile storage.

Types of system failure:

1. Transaction Failure: Occurs when a transaction cannot proceed due to an error or violation of constraints.
2. System Crash: A sudden failure of the database system, usually caused by hardware or software issues.
3. Disk Failure: Refers to the failure of storage devices, resulting in the loss of data.
4. Media Failure: Involves the corruption or loss of data due to factors like disk errors, file system corruption, or natural disasters

Q-14: Define lock with example. Discuss different types of lock used in DBMS. [pre-3:1_2021]

Answer:

A lock in DBMS (Database Management System) is a synchronization mechanism used to control concurrent access to shared resources in a database. It ensures that only one transaction can modify or read a resource at a time, preventing conflicts and maintaining data consistency.

Example: Let's consider a scenario where two users, A and B, simultaneously try to update a balance in a bank account. To maintain consistency, a lock is used to ensure that only one user

can modify the balance at a time. User A obtains a lock on the account, modifies the balance, and releases the lock. Then, user B acquires the lock and performs the update.

Different types of locks in DBMS:

1. Shared (Read) Lock: Allows concurrent read access to a resource but prevents write access by other transactions.
2. Exclusive (Write) Lock: Prevents other transactions from both read and write access to a resource.

Q-15: Discuss two phase locking protocol along with its required diagram. [pre-3:1_2021]

Answer:

The Two-Phase Locking (2PL) protocol is a concurrency control mechanism that ensures serializability of transactions. It consists of two phases: the growing phase and the shrinking phase.

During the growing phase, transactions acquire locks on resources and hold them until they no longer request any new locks. In this phase, a transaction can only acquire new locks but cannot release any locks.

The shrinking phase starts when a transaction releases locks. Once a transaction releases a lock, it cannot acquire any new locks. This phase continues until the transaction releases all its locks.

The 2PL protocol ensures serializability, prevents conflicts, and ensures a consistent view of the database. Here's a diagram illustrating the two phases:

🔒 Growing Phase: T1 acquires locks --> T2 acquires locks --> T3 acquires locks 🔓
Shrinking Phase: T1 releases locks --> T2 releases locks --> T3 releases locks

Q-16: What do you mean by fail-stop assumption? Classify failure occurred in DBMS and describe each of them. [pre-3:1_2021]

Answer:

Fail-stop assumption refers to the assumption that a failed component in a distributed system stops processing and does not corrupt existing data.

Failures in DBMS can be classified as:

1. Transaction Failure: A transaction fails to complete due to an error or violation of constraints. It can be classified further as system errors or application errors.
2. System Failure: This includes hardware or software failures, such as disk crashes, power outages, or network failures, that result in the entire system becoming unavailable.
3. Media Failure: Refers to the permanent loss or corruption of data due to factors like disk failures, file system corruption, or natural disasters.
4. Disk Failure: Specifically pertains to the failure of storage devices, resulting in the loss of data stored on those devices.

Q-17:Discuss different types of log-based recovery mechanism. [pre-3:1_2021]

Answer:

Different types of log-based recovery mechanisms include:

1. Undo Logging: The system maintains an undo log, which records the before images of the data modified by a transaction. During recovery, the transactions are undone using the undo log in reverse order to bring the database to a consistent state.
2. Redo Logging: The system maintains a redo log, which records the after images of the data modified by a transaction. During recovery, the redo log is applied to the database to reapply the changes made by committed transactions.
3. Checkpointing: Periodically, the system creates a checkpoint that records the state of the database at that point. During recovery, the checkpoint is used to reduce the amount of work needed for undo and redo operations.

Q-18:What is starvation? Why does it happen in DBMS? Give an example. .[pre-3:1_2021]

Answer:

Starvation refers to a situation where a transaction or process is unable to proceed or make progress due to insufficient resources or the presence of higher-priority transactions/processes.

In DBMS, starvation can occur when a transaction is continually denied access to a resource because other transactions constantly acquire and hold the required resources. This can lead to indefinite delays for the starved transaction, negatively impacting the system's performance and response times.

For example, consider a scenario where multiple transactions require exclusive locks on a resource, and a low-priority transaction keeps getting preempted by higher-priority transactions. As a result, the low-priority transaction remains in a waiting state indefinitely, unable to acquire the necessary locks and complete its execution.

Q-19:What is meant by concurrent execution of database transaction in a multi user system? Discuss why concurrency control is needed in a database transaction. [pre-3:1_2021]

Answer:

Concurrent execution of database transactions in a multi-user system refers to the situation where multiple transactions are executing simultaneously and accessing shared resources concurrently. This concurrency can improve system performance and throughput.

Concurrency control is needed in database transactions to ensure the following:

1. Data Integrity: Concurrent transactions should not interfere with each other in a way that violates data integrity constraints or produces inconsistent results.
2. Consistency: The execution order of transactions should maintain the consistency of the database and adhere to predefined rules and constraints.
3. Isolation: Transactions should be executed as if they were executed sequentially, without being aware of the concurrent execution of other transactions.

Concurrency control mechanisms, such as locks, timestamps, or optimistic techniques, are employed to coordinate and manage concurrent access to shared resources, ensuring data integrity, consistency, and isolation. 😜💻🔒

Relational Database Design:

Q-01: Discuss four important features of good relational database design.

Answer:

Four important features of good relational database design are:

1. Data Integrity: A well-designed relational database ensures data integrity by enforcing constraints and rules that maintain the accuracy, consistency, and validity of the data. This includes defining primary and foreign key relationships, implementing referential integrity, and specifying data types and constraints on attributes.

2. Normalization: Good database design involves normalizing the data to eliminate redundancy and dependency issues. Normalization aims to organize the data into logical and efficient structures, reducing data duplication and improving data consistency. By following normalization rules, such as the normalization forms (e.g., First Normal Form, Second Normal Form), a database can avoid update anomalies and optimize query performance.

3. Scalability: A good relational database design accommodates future growth and scalability requirements. This includes appropriately defining the structure and relationships between tables to support the addition of new entities and attributes. Scalability can also be achieved by

considering performance optimization techniques, such as indexing, partitioning, and denormalization.

4. Efficiency and Performance: Efficient database design focuses on optimizing database operations to ensure smooth and fast data retrieval and manipulation. This involves carefully designing indexes, choosing appropriate data types and sizes, and considering query optimization techniques. Well-structured database schemas and efficient query plans contribute to better overall performance.

In summary, a good relational database design prioritizes data integrity, normalization, scalability, and efficiency to ensure a robust, reliable, and high-performing database system. 😊💾

MD RAIHAN ALI_[63]
CSE_02
Faculty of Engineering & Technology
University of Dhaka (NITER)