Module-1

Purpose of Database System:

A database is a structured collection of organized data that is stored and managed in a way that makes it easy to access and manage. In this data stored in tables,which are made up of rows and columns. Each row represents a single record and each columns represents a single piece of information about that record. Its provids user with tools for creating and modifying database as well as for inserting, updating, deleting and querying data.

Characteristics of DBMS

- •It uses a digital repository established on a server to store and manage the information.
- •It can provide a clear and logical view of the process that manipulates data.
- •DBMS contains automatic backup and recovery procedures.
- •It contains ACID properties which maintain data in a healthy state in case of failure.
- •It can reduce the complex relationship between data.
- •It is used to support manipulation and processing of data.
- •It is used to provide security of data.
- •It can view the database from different viewpoints according to the requirements of the user.

Advantages of DBMS:

- •**Controls database redundancy:**It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- •**Data sharing:**In DBMS, the authorized users of an organization can share the data among multiple users.
- •**Easily Maintenance:**It can be easily maintainable due to the centralized nature of the database system.
- •**Reduce time:**It reduces development time and maintenance need.
- •**Backup:**It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- •**multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- •**Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- •**Size:** It occupies a large space of disks and large memory to run them efficiently.
- •**Complexity:** Database system creates additional complexity and requirements.
- •**Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

Views of data:

Views of data refer to different perspectives or representations of the underlying data in a database. In a database system, data can be viewed and accessed in various ways based on the specific needs and permissions of different users or applications. These views provide a tailored and controlled access to the data, hiding certain information or presenting data in a more meaningful and relevant manner.

There are views of data in a database system:

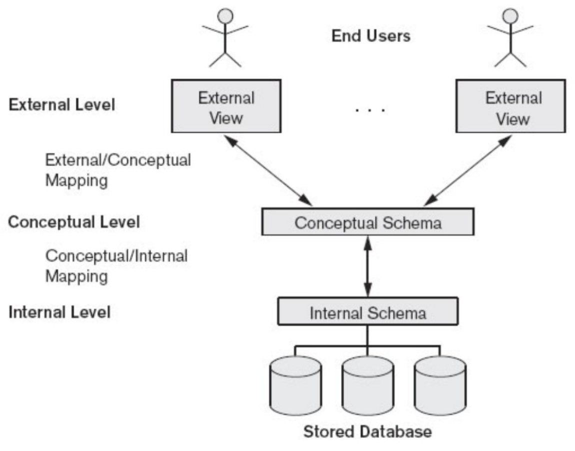
1. External View (User View): The external view represents the data from the perspective of

individual users or specific applications. Each user or application may have a different view of the data, showing only the relevant information required for their tasks. External views are customized to meet the unique requirements and preferences of users and provide an abstraction of the overall data structure.
For example, in a university database, the external view for students may include information such as course schedules, grades, and personal details, while the external view for professors may include information related to teaching assignments, research, and student evaluations.

2. Conceptual View (Logical View): Conceptual Level (Logical Data Model): The conceptual level, also known as the logical level, represents the overall logical structure of the database. It defines the relationships between various entities, attributes, and tables without focusing on the specific data access requirements of individual users. The conceptual level provides a high-level understanding of the database's organization and helps in database design and management.

   For example, in the same university database, the conceptual view would define the relationships between entities like students, courses, and professors, without delving into the specific data each user is allowed to access.

3. Internal Level (Physical Data Storage): The internal level, also known as the physical level, represents the physical storage and access methods of the data. It deals with how data is stored on storage devices, such as hard drives, solid-state drives, or magnetic tapes. The internal level is concerned with data storage structures, indexing techniques, and data retrieval algorithms.



Data Models:

Data models are abstract representations that describe how data is organized, stored, and related in a database system. They serve as blueprints for designing databases and provide a clear and structured way to understand the relationships and interactions between different data elements. Data models facilitate communication between stakeholders, such as database designers, developers, and end-users, and ensure that the database meets the requirements of the application or business process.

There are several types of data models, with the most widely used ones being:

1. Hierarchical Data Model: In the hierarchical data model, data is organized in a tree-like structure with a single parent and multiple child nodes. Each child node can have only one parent, and the relationship forms a hierarchy. This model was prevalent in early database systems but is not commonly used in modern databases due to its limited flexibility.
2. Network Data Model: The network data model is an extension of the hierarchical model and allows more complex relationships between data elements. It uses a graph structure, where each record can have multiple parent and child nodes, enabling more flexibility in representing data relationships. However, like the hierarchical model, the network data model is not as popular as the relational data model.
3. Entity-Relationship Model (E-R Model): The entity-relationship (E-R) model is a widely used data modeling technique based on the concepts of entities, attributes, and relationships. It

represents data as entities (real-world objects or concepts), attributes (properties of entities), and relationships (associations between entities). The E-R model is the foundation for creating E-R diagrams, which visually represent the structure of the database.

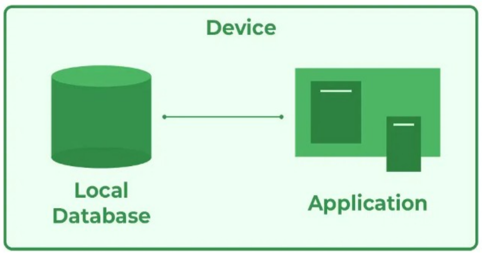
4. Relational Data Model: The relational data model is the most prevalent data model used in modern database systems. It organizes data into tables, where each table represents an entity, and each row in the table represents a specific record (instance of the entity). The columns in the table represent attributes (properties) of the entity. Relationships between entities are established using primary keys and foreign keys.
5. Object-Oriented Data Model: The object-oriented data model is based on the principles of object-oriented programming. It represents data as objects, which encapsulate both data and behavior. Objects can inherit properties and methods from other objects, enabling the creation of complex data structures. This model is mainly used in object-oriented databases.
6. Object-Relational Data Model: The object-relational data model combines features from the relational and object-oriented data models. It extends the relational model to support object-oriented concepts like inheritance, encapsulation, and methods. This model is used in object-relational databases to handle complex data types and relationships.

Database System Architecture:

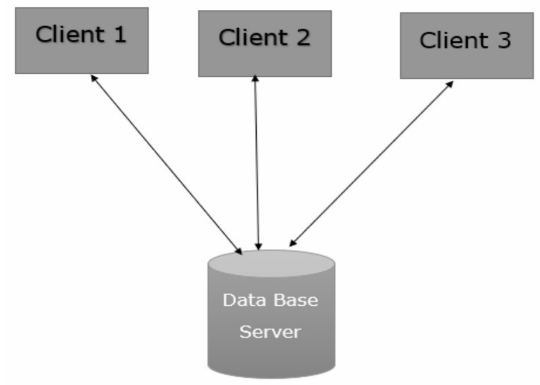
Database system architecture refers to the overall structure and components of a database management system (DBMS) that enables the storage, retrieval, and manipulation of data in a database. The architecture of a database system is designed to provide an efficient and reliable way to manage data and support various applications and users.

Database system architectures can also be classified by the number of tiers they have. A tier is a logical grouping of components that perform a similar function. The most common types of database system architectures are:

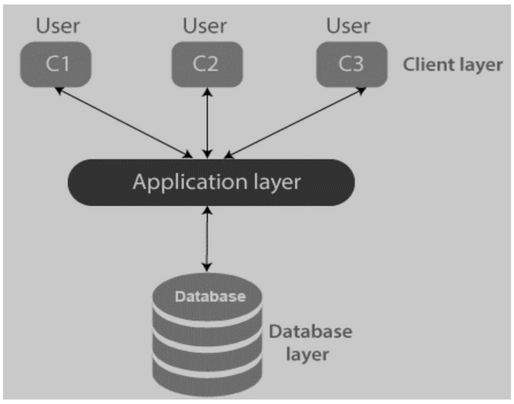
Single-tier architecture: In a single-tier architecture, all of the database components are located on the same tier. This is the simplest type of architecture, but it is also the least scalable and performant.



Two-tier architecture: In a two-tier architecture, the database components are divided into two tiers: a client tier and a server tier. The client tier is responsible for displaying the user interface and handling user requests. The server tier is responsible for storing and managing the database. This type of architecture is more scalable and performant than a single-tier architecture, but it can be more complex to implement and manage.



Three-tier architecture: In a three-tier architecture, the database components are divided into three tiers: a presentation tier, an application tier, and a data tier. The presentation tier is responsible for displaying the user interface and handling user requests. The application tier is responsible for processing business logic and interacting with the data tier. The data tier is responsible for storing and managing the database. This type of architecture is the most scalable and performant, but it is also the most complex to implement and manage.



Database users and Administrator:

Database users and administrators play critical roles in the management and utilization of a database system:

1. **Database Users:** Database users are individuals or applications that interact with the database to perform various operations, such as querying, inserting, updating, and deleting data. They can be categorized into different types based on their roles and responsibilities:

- End Users: These are regular users who access the database to retrieve information or perform simple operations. They may include employees, customers, students, or any other individuals using applications that access the database.
- Application Users: These are users who interact with the database through software applications. Application users include programmers and developers who create and maintain applications that use the database to support specific tasks or business processes.
- Power Users: Power users are advanced users who have a deeper understanding of the database and can perform complex queries and analyses. They may include analysts, data scientists, or specialized users who require more in-depth access to the data.
- Executive Users: Executive users are senior management or decision-makers who use the database to gain insights and make strategic decisions based on the data.

2. **Database Administrator (DBA):** The database administrator, often referred to as DBA, is responsible for managing and maintaining the database system. They play a crucial role in ensuring the smooth operation, security, and integrity of the database. The DBA's responsibilities may include:

- Database Installation and Configuration: Installing and configuring the database management system software on servers or computing environments.
- Security Management: Managing user access and permissions to ensure data security and prevent unauthorized access to sensitive information.
- Backup and Recovery: Implementing backup and recovery strategies to protect against data loss and system failures.
- Performance Tuning: Monitoring and optimizing database performance to ensure efficient data retrieval and processing.
- Database Design and Maintenance: Designing and maintaining the database structure, including tables, indexes, and relationships, based on application requirements.
- Data Integrity and Quality: Ensuring data integrity and data quality by enforcing constraints and validating data entered into the database.
- Database Upgrades and Patching: Performing upgrades and applying patches to keep the

database software up to date and secure.
- Troubleshooting: Identifying and resolving issues related to database performance, data corruption, or system errors.
- Capacity Planning: Estimating future storage requirements and planning for database growth.

Entity:

An entity in DBMS (Database management System) is a real-world thing or a real-world object which is distinguishable from other objects in the real world. For example, a car is an entity. An attribute of an entity gives us information about the characteristic features of an entity. For example, a black car entity has an attribute called color which has a value black.

Relationship model (E-R model):

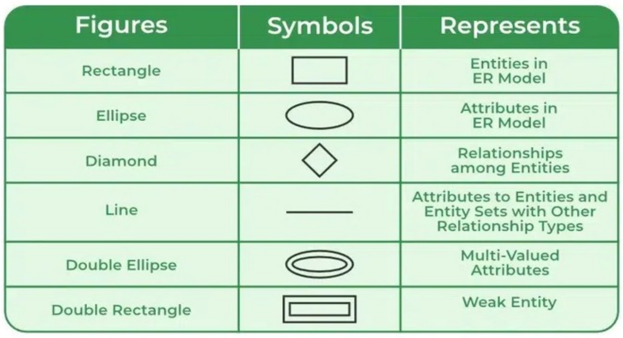
The Entity-Relationship (E-R) model is a widely used data modeling technique used to design databases and represent the logical structure of a database system. It was proposed by Peter Chen in 1976 as a way to visualize and describe the relationships between different entities in a database. The E-R model is based on the concept of entities, attributes, and relationships.

Key components of the E-R model include:

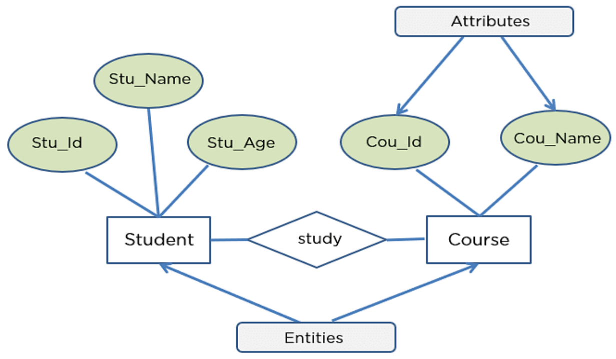
1. Entity: An entity is a distinct real-world object, concept, or thing that can be uniquely identified and is represented by a rectangle in an E-R diagram. Each entity has attributes that describe its properties or characteristics. For example, in a university database, entities could include "Student," "Course," and "Professor."
2. Attribute: Attributes are the properties or characteristics of an entity. They provide additional information about the entity. Attributes are represented within the rectangles corresponding to the entities in the E-R diagram. For example, attributes of the "Student" entity could include "Student ID," "Name," and "Date of Birth."
3. Relationship: A relationship is an association between two or more entities. It represents how entities are related or connected to each other. Relationships are depicted using lines connecting the related entities in an E-R diagram. Each relationship has a name that describes the nature of the association. For example, a "Takes" relationship between the "Student" and "Course" entities represents that a student takes a particular course.
4. Cardinality: Cardinality defines the number of instances or occurrences of one entity that can be associated with the number of instances of another entity through a relationship. It specifies the minimum and maximum number of relationships allowed between entities. Cardinality can be one-to-one (1:1), one-to-many (1:N), or many-to-many (N:M). For instance, a student can take many courses (one-to-many) in a university database.
5. Primary Key: A primary key is an attribute or a combination of attributes that uniquely identifies each instance or row of an entity in the database. It is used to ensure that each entity instance has a unique identity. Primary keys are crucial for establishing relationships between entities through foreign keys.
6. Foreign Key: A foreign key is an attribute in one entity that refers to the primary key of another entity, establishing a relationship between the two entities. Foreign keys are used to enforce referential integrity and maintain consistency in the database.

E-R Diagrams:

An Entity-Relationship (E-R) diagram, also known as an E-R model diagram, is a visual representation of the Entity-Relationship (E-R) model, which is a popular data modeling technique used to design databases. E-R diagrams help in understanding and communicating the relationships and interactions between different entities in a database system.



Figures	Symbols	Represents
Rectangle	▭	Entities in ER Model
Ellipse	⬭	Attributes in ER Model
Diamond	◇	Relationships among Entities
Line	—	Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse	◎	Multi-Valued Attributes
Double Rectangle	▣	Weak Entity



Cardinality relation:

Cardinality relation refers to the way tables in a relational database are associated with each other. It describes the number of related records or rows in one table that can be linked to the records in another table through a specific relationship. Cardinality helps define the type and strength of the relationship between tables and is crucial for establishing referential integrity and optimizing database performance.

Types of cardinality:

1. One-to-One (1:1) Cardinality: In a one-to-one cardinality relationship, one instance of an entity is associated with at most one instance of another entity, and vice versa. This means that each instance of one entity is related to only one instance of the other entity, and each instance of the other entity is related to only one instance of the first entity. One-to-one cardinality is relatively rare in relational databases and is typically used for situations where two entities have a unique and strict relationship.

Example: In a database representing a person and their passport, each person can have only one passport, and each passport is associated with only one person.

2. One-to-Many (1:N) Cardinality: In a one-to-many cardinality relationship, one instance of an entity can be associated with multiple instances of another entity, but each instance of the other entity is associated with only one instance of the first entity. This type of cardinality is common and represents a hierarchical or parent-child relationship.

Example: In a database representing a department and its employees, one department can have many employees, but each employee belongs to only one department.

3. Many-to-Many (N:M) Cardinality: In a many-to-many cardinality relationship, multiple instances of one entity can be associated with multiple instances of another entity, and vice versa. This means that each instance of one entity can be related to many instances of the other entity, and each instance of the other entity can be related to many instances of the first entity. Many-to-many cardinality requires the use of an intermediate or junction table to establish the relationship between the two entities.

Example: In a database representing students and courses, each student can enroll in multiple courses, and each course can have multiple students. To represent this many-to-many relationship, a separate table (e.g., Enrollment table) is used to link students with courses.

Introduction to relational databases:

Relational databases are a type of database management system (DBMS) that organizes and stores data in a structured manner, using a tabular format known as tables. These databases are based on the relational model, which was introduced by Dr. E.F. Codd in 1970. In a relational database, data is represented using rows and columns, where each row represents a record (also known as a tuple or row instance), and each column represents an attribute or field of the record.

Key features of relational databases include:

1. Tables: The primary data structure in a relational database is the table, also called a relation. Each table consists of rows (records) and columns (attributes). Tables are used to store and organize data in a structured manner, with each row representing a unique record, and each column representing a specific attribute of the record.
2. Keys: Relational databases use keys to uniquely identify records within a table and establish relationships between tables. The primary key is a unique identifier for each record in a table, ensuring that each record has a distinct identity. Foreign keys are used to create relationships between tables by linking the primary key of one table to the corresponding attribute in another table.
3. Relationships: Relational databases can represent complex relationships between different sets of data using foreign keys. These relationships allow data to be spread across multiple related tables while maintaining data integrity and ensuring referential integrity.
4. (Structured Query Language): is the standard language used to interact with relational databases. It provides a set of commands for querying, inserting, updating, and deleting data in the database. allows users and applications to retrieve and manipulate data stored in tables.

Advantages of Relational Databases:

- Data Integrity: Relational databases enforce constraints and rules to maintain data integrity and ensure that the data is accurate and consistent.
- Flexibility: The structure of a relational database can be easily modified to accommodate changes in data requirements without affecting existing data or applications.
- Data Security: Relational databases support access controls and permissions to restrict access to sensitive data, ensuring data security.
- Data Independence: The relational model allows for data independence, separating the physical storage of data from the logical view, making it easier to manage and maintain the database.

Dr EF Codd rules:

Dr. Edgar F. Codd, often referred to as E.F. Codd, was a British computer scientist and mathematician who is widely regarded as the father of the relational database model. In 1970, he introduced a set of principles known as "Codd's 12 Rules" to define the characteristics of a true relational database management system (RDBMS). These rules laid the foundation for modern relational databases and served as a benchmark for evaluating the compliance of database systems with the relational model. Over time, the original 12 rules were condensed into 13, and they are as follows:

1. Foundation Rule: The database must be in the relational form . So the system can handle the database through its relational capabilities.
2. Information Rule: A database contains various information, and this information must be stored in each cell of a table in the form of rows and columns. .
3. Guaranteed Access Rule: Each data value in the database should be accessible through a combination of table name, primary key value, and column name.
4. Systematic Treatment of Null Values: The DBMS must provide systematic ways to handle missing or unknown data values, known as NULLs, to avoid ambiguity and inconsistency.
5. Dynamic Online Catalog Based on the Relational Model: The database's structure and schema (metadata) should be stored in tables and made accessible through the same query language used for data manipulation, promoting a unified approach to managing database information.

6. Comprehensive Data Sublanguage Rule: The DBMS must support a comprehensive data sublanguage that allows users to define, manipulate, and retrieve data without exposing the underlying physical storage details.
7. View Updating Rule: All views that are theoretically updatable must also be updatable by the system.
8. High-Level Insert, Update, and Delete Rule: The DBMS should support high-level operations (insert, update, delete) that allow users to modify data without having to specify detailed low-level procedures.
9. Physical Data Independence: Changes to the physical storage structure of the database should not affect the logical schema or application programs.
10. Logical Data Independence: Changes to the logical schema should not affect the existing applications or user views.
11. Integrity Independence: Integrity constraints should be defined separately from application programs and stored in the data dictionary to ensure consistency across the entire database.
12. Distribution Independence: The DBMS should provide distribution transparency, allowing data to be distributed across multiple locations without affecting the user's view of the data.
13. Non-Subversion Rule: The DBMS must not allow users to bypass security and integrity constraints to access or modify data.

Database Languages:

Database languages are specialized programming languages used to interact with databases and perform various operations, such as querying, inserting, updating, and deleting data. These languages are designed to communicate with the underlying database management system (DBMS) and allow users, applications, and administrators to access and manipulate data stored in the database.

fundamentals:

(Structured Query Language) is a domain-specific language used to manage and manipulate relational databases. It provides a standardized way to interact with database management systems (DBMS) and perform various operations on the data, such as querying, inserting, updating, and deleting records. is a fundamental skill for anyone working with databases and is widely used in database development, data analysis, and data management.

Here are some fundamentals to get you started:

1. Data Retrieval (SELECT): The SELECT statement is used to retrieve data from one or more database tables. It allows you to specify the columns you want to retrieve, the table you want to query, and any filtering or sorting criteria. The basic syntax is as follows:

Syntax-:
```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:
```
SELECT first_name, last_name, age
FROM employees
WHERE department = 'IT'
ORDER BY last_name;
```

2. Data Insertion (INSERT): The INSERT statement is used to add new records or rows to a database table. It allows you to specify the values for each column in the new record. The basic syntax is as follows:

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

Example:
```
INSERT INTO employees (first_name, last_name, age, department)
VALUES ('John', 'Doe', 30, 'HR');
```

3. Data Updating (UPDATE): The UPDATE statement is used to modify existing records in a database table. It allows you to change the values of specific columns for selected rows. The

basic syntax is as follows:

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

Example:
```
UPDATE employees
SET department = 'Finance'
WHERE last_name = 'Smith';
```

4. Data Deletion (DELETE): The DELETE statement is used to remove records from a database table. It allows you to specify a condition to identify the rows to be deleted. The basic syntax is as follows:

Syntax:
```
DELETE FROM table_name
WHERE condition;
```

Example:
```
DELETE FROM employees
WHERE department = 'IT';
```

5. Data Filtering (WHERE): The WHERE clause is used to filter records based on specified conditions. It allows you to narrow down the result set and retrieve only the data that meets the specified criteria.

Example:
```
SELECT product_name, price
FROM products
WHERE price > 50;
```

6. Data Sorting (ORDER BY): The ORDER BY clause is used to sort the result set based on one or more columns in ascending or descending order.

Example:
```
SELECT product_name, price
FROM products
ORDER BY price DESC;
```

Sub languages of :

(Structured Query Language) is a comprehensive language with various sub-languages or categories of commands that serve different purposes in database management. The main sub-languages of include:

1. Data Definition Language (DDL): DDL is used to define and manage the structure of the database schema. It includes commands for creating, altering, and dropping database objects such as tables, views, indexes, and constraints.

Common DDL commands include:

- CREATE: Creates a new database object, such as a table or view.
- ALTER: Modifies the structure of an existing database object.
- DROP: Deletes a database object, such as a table or index.
- TRUNCATE: Removes all data from a table without deleting the table structure.
- COMMENT: Adds comments to the database objects for documentation purposes.

2. Data Manipulation Language (DML): DML is used to manipulate and retrieve data stored in the database. It includes commands for inserting, updating, deleting, and querying data from the tables.

Common DML commands include:

- SELECT: Retrieves data from one or more tables using queries.
- INSERT: Adds new records or rows into a table.
- UPDATE: Modifies existing records or rows in a table.

- DELETE: Removes records or rows from a table.
- MERGE: Performs insert, update, or delete operations based on a condition.

3. Data Control Language (DCL): DCL is used to control access and permissions to the database objects. It includes commands for granting or revoking privileges to users and controlling their level of access to the data.

Common DCL commands include:

- GRANT: Provides users with specific privileges or permissions to perform operations on the database.
- REVOKE: Removes previously granted privileges from users.
- DENY: Explicitly denies certain permissions to users.

4. Data Query Language (DQL): Some databases consider DQL as a subset of DML, but it is often treated as a separate sub-language. DQL is used exclusively for querying and retrieving data from the database.

Common DQL command:

- SELECT: Retrieves data from one or more tables using queries.

5. Transaction Control Language (TCL): TCL is used to manage transactions in the database. Transactions are sequences of one or more database operations that are treated as a single unit of work.

Common TCL commands include:

- COMMIT: Confirms the changes made in the current transaction and makes them permanent.
- ROLLBACK: Reverts the changes made in the current transaction and restores the data to its previous state.
- SAVEPOINT: Sets a named point within a transaction to which you can later roll back.

Logical Operator:

Range Searching:

Range searching refers to the process of searching for data that falls within a specific range of values. It is a common operation used to retrieve data that meets certain criteria based on a range of numeric or date values. Range searching is often performed using the WHERE clause in a SELECT statement to filter data based on a specified range.

In , you can use various comparison operators for range searching:

1. Greater Than (>): Retrieves data that is greater than a specified value.
2. Greater Than or Equal To (>=): Retrieves data that is greater than or equal to a specified value.
3. Less Than (<): Retrieves data that is less than a specified value.
4. Less Than or Equal To (<=): Retrieves data that is less than or equal to a specified value.
5. BETWEEN: Retrieves data that falls within a specified range, including the boundary values.

Here's an example of using the BETWEEN operator for range searching:

Suppose you have a table called "products" with a column "price" representing the price of each product. To retrieve products with prices between $10 and $50, you can use the following query:

```
SELECT product_name, price
FROM products
```

```
WHERE price BETWEEN 10 AND 50;
```

This query will return all products with prices that are greater than or equal to $10 and less than or equal to $50.

You can also use the combination of greater than and less than operators for more complex range searches:

```
SELECT order_id, order_date, total_amount
FROM orders
WHERE total_amount > 1000 AND total_amount <= 2000;
```

This query will return orders with a total amount greater than $1000 and less than or equal to $2000.

Pattern Matching:

Pattern matching refers to the process of searching for data that matches a specific pattern or sequence of characters within a text or string field. It allows you to find data based on partial or incomplete information, making it a powerful tool for retrieving relevant information from the database.

Pattern matching is commonly used in queries, especially with the use of the LIKE operator and wildcard characters. The most commonly used wildcard characters are:

1. `%` (Percent Sign): It represents zero, one, or multiple characters. For example, `**'%apple%'**` would match any string that contains the word "apple" anywhere within it.

2. `_` (Underscore): It represents a single character. For example, `'ha_ll_'` would match "hello," "hall," and any other four-letter word that starts with "ha" and ends with "ll."

Here are some examples of pattern matching in :

1. Find all customers whose names start with "J":

```
SELECT * FROM customers
WHERE customer_name LIKE 'J%';
```

2. Find all products whose names contain the word "phone":

```
SELECT * FROM products
WHERE product_name LIKE '%phone%';
```

3. Find all email addresses that end with "example.com":

```
SELECT * FROM emails
WHERE email_address LIKE '%@example.com';
```

Order by clause:

The ORDER BY clause is a clause used in database management systems (DBMS) to sort the result set of a query based on one or more specified columns. It allows you to control the order in which the rows of the query result are presented, making it easier to analyze and understand the data.

The basic syntax of the ORDER BY clause in is as follows:

SELECT column1, column2, ...
FROM table_name
WHERE condition
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;

- `column1, column2, ...`: The columns by which you want to sort the result set. You can specify multiple columns, and the sorting order will be applied in the order they are listed.
- `ASC`: Stands for "ascending," which means the result set will be sorted in ascending order (from smallest to largest) based on the specified columns. This is the default sorting order if ASC or DESC is not explicitly specified.
- `DESC`: Stands for "descending," which means the result set will be sorted in descending order (from largest to smallest) based on the specified columns.

Examples:

1. Sorting in Ascending Order:

SELECT product_name, price
FROM products
ORDER BY price ASC;

This query retrieves the product_name and price columns from the products table and sorts the result set based on the price column in ascending order (from the lowest price to the highest price).

2. Sorting in Descending Order:

SELECT employee_name, hire_date
FROM employees
ORDER BY hire_date DESC;

This query retrieves the employee_name and hire_date columns from the employees table and sorts the result set based on the hire_date column in descending order (from the most recent hire date to the oldest).

You can use the ORDER BY clause in combination with other clauses, such as WHERE and GROUP BY, to perform more complex data analysis and presentation in your queries. Sorting the result set with ORDER BY can be helpful in presenting data in a meaningful way, especially when dealing with large datasets.

Group By Clause

The GROUP BY clause is a clause used in database management systems (DBMS) to group rows based on the values in one or more specified columns. It allows you to perform aggregate functions on groups of data, such as calculating sums, averages, counts, or other statistics within each group.

The basic syntax of the GROUP BY clause in is as follows:

SELECT column1, column2, ..., aggregate_function(columnX)
FROM table_name
WHERE condition
GROUP BY column1, column2, ...;

- `column1, column2, ...`: The columns by which you want to group the data. You can specify one or multiple columns for grouping.
- `aggregate_function(columnX)`: An aggregate function applied to a specific column (columnX) within each group. Common aggregate functions include SUM, AVG, COUNT, MAX, MIN, etc.

When you use the GROUP BY clause, the result set will be divided into groups based on the unique combinations of values in the specified columns. Then, the aggregate functions will be applied to each group separately to calculate the desired summary information.

Examples:

1. Grouping and Calculating Sum:

```
SELECT department, SUM(salary) AS total_salary
FROM employees
GROUP BY department;
```

This query groups the employees by their department and calculates the total salary for each department using the SUM aggregate function.

2. Grouping and Calculating Average:

```
SELECT category, AVG(price) AS average_price
FROM products
GROUP BY category;
```

This query groups the products by their category and calculates the average price for each category using the AVG aggregate function.

3. Grouping with Multiple Columns:

```
SELECT department, location, COUNT(*) AS num_employees
FROM employees
GROUP BY department, location;
```

Module – 2

Normalization:

Normalization is a process in database design that aims to organize and structure relational databases efficiently, minimizing redundancy and data anomalies. The goal of normalization is to eliminate data duplication and create well-structured tables that adhere to certain rules, called normal forms. Normalization helps maintain data integrity and reduces the likelihood of data inconsistencies, update anomalies, and other issues that may arise in a poorly designed database.
The process of normalization involves breaking down a large table into multiple smaller tables and establishing relationships between them through the use of keys. There are several normal forms, each addressing specific types of data redundancy and anomalies.

Functional Dependencies:

Functional dependencies are a core concept in database management systems and are closely related to the process of normalization. A functional dependency is a relationship between attributes (columns) in a relational database table, where the value of one attribute uniquely determines the value of another attribute.

In simpler terms, if attribute A functionally determines attribute B, it means that for each unique value of A, there is only one corresponding value of B in the table.
Functional dependencies are represented using an arrow symbol (->), where the attribute(s) on the left side determine the attribute(s) on the right side. The general notation for a functional dependency is:
A -> B
Where A is the determining attribute (or a set of attributes) and B is the determined attribute.

For example, consider a table called "Employees" with the following attributes: employee_id, employee_name, department, and salary. If we assume that employee_id uniquely identifies employee_name (each employee_id has only one associated employee_name), we can express this functional dependency as:
employee_id -> employee_name
This means that given an employee_id, we can determine the corresponding employee_name without ambiguity.
Functional dependencies are essential in database design and normalization, as they help identify potential data redundancy and anomalies. By identifying functional dependencies, we can determine the candidate keys (minimal set of attributes that uniquely identify each row) and eliminate partial dependencies, which leads to a more efficient and well-structured database design.

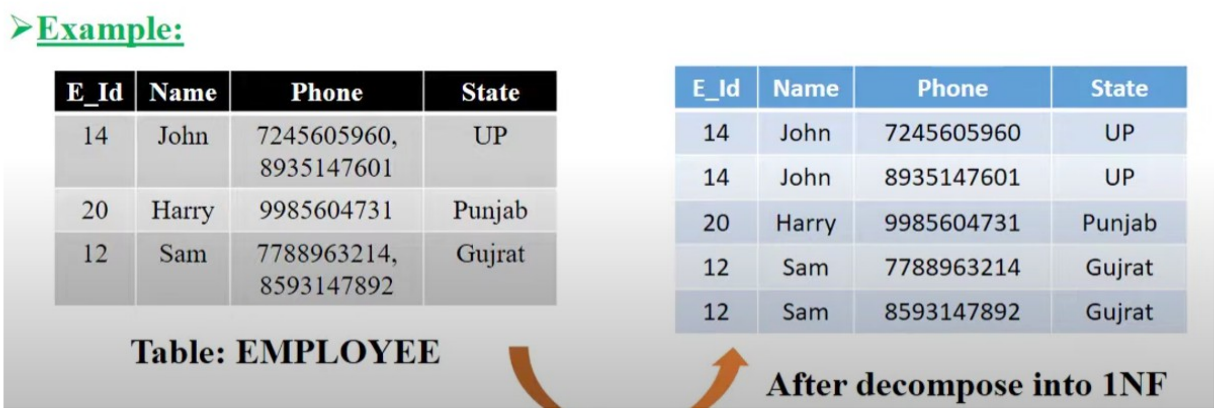
Non-loss Decomposition:

Non-loss decomposition, also known as lossless decomposition, is a concept in database normalization that ensures the preservation of functional dependencies and data integrity when splitting a relation (table) into multiple smaller relations. The goal of non-loss decomposition is to avoid losing any information during the decomposition process, so that the original data can be reconstructed from the smaller relations without any loss or ambiguity.
In the context of normalization, a relation is said to be in non-loss decomposition if:

1. The original relation can be reconstructed entirely by joining the smaller decomposed relations back together.
2. The functional dependencies that held in the original relation are preserved in the smaller relations.

Types of normalization:

1. First Normal Form (1NF):
First Normal Form deals with the most basic level of normalization. To satisfy 1NF, a table must meet the following conditions:
- Each cell (entry) in the table must hold a single, atomic value (no repeating groups or arrays of values).
- Each column in the table must have a unique name.
- Each row in the table must have a unique identifier, often referred to as the primary key, which uniquely identifies each row in the table.


➢Example:

E_Id	Name	Phone	State
14	John	7245605960, 8935147601	UP
20	Harry	9985604731	Punjab
12	Sam	7788963214, 8593147892	Gujrat

Table: EMPLOYEE

E_Id	Name	Phone	State
14	John	7245605960	UP
14	John	8935147601	UP
20	Harry	9985604731	Punjab
12	Sam	7788963214	Gujrat
12	Sam	8593147892	Gujrat

After decompose into 1NF

2. Second Normal Form (2NF):
Second Normal Form builds upon 1NF and addresses partial dependencies. To satisfy 2NF, a table must first be in 1NF, and it must not have partial dependencies. A partial dependency occurs when a non-key

attribute depends on only a part of the primary key.

Example of a table that violates 2NF:

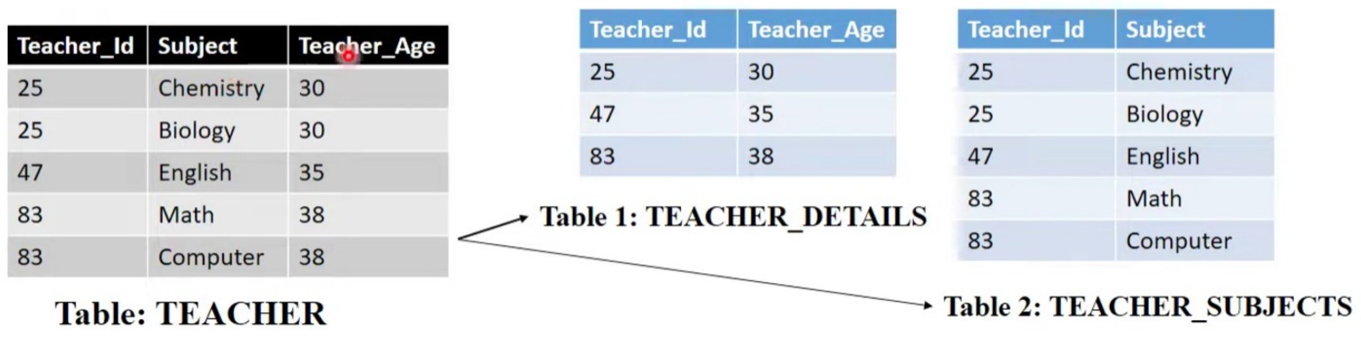


Table: TEACHER

Teacher_Id	Subject	Teacher_Age
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

Table 1: TEACHER_DETAILS

Teacher_Id	Teacher_Age
25	30
47	35
83	38

Table 2: TEACHER_SUBJECTS

Teacher_Id	Subject
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

3. Third Normal Form (3NF):
Third Normal Form builds upon 2NF and addresses transitive dependencies. To satisfy 3NF, a table must first be in 2NF, and it must not have transitive dependencies. A transitive dependency occurs when a non-key attribute depends on another non-key attribute rather than the primary key directly.

Example of a table that violates 3NF:

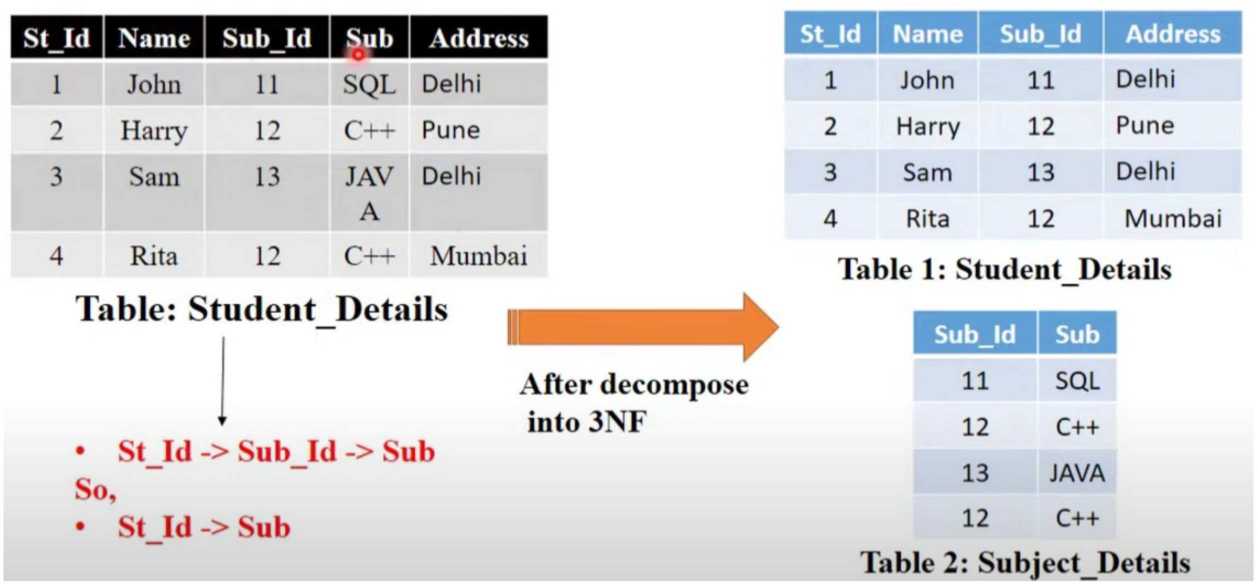


Table: Student_Details

St_Id	Name	Sub_Id	Sub	Address
1	John	11	SQL	Delhi
2	Harry	12	C++	Pune
3	Sam	13	JAVA	Delhi
4	Rita	12	C++	Mumbai

- St_Id -> Sub_Id -> Sub

So,

- St_Id -> Sub

After decompose
into 3NF

Table 1: Student_Details

St_Id	Name	Sub_Id	Address
1	John	11	Delhi
2	Harry	12	Pune
3	Sam	13	Delhi
4	Rita	12	Mumbai

Table 2: Subject_Details

Sub_Id	Sub
11	SQL
12	C++
13	JAVA
12	C++

Dependency Preservation:

Dependency preservation is a critical property in database normalization, specifically in the context of decomposing a relation (table) into multiple smaller relations while ensuring that the functional dependencies present in the original relation are preserved in the decomposed relations. The goal of dependency preservation is to maintain the relationships between attributes and prevent the loss of information during the decomposition process.

Boyce/Codd Normal Form:

Boyce-Codd Normal Form (BCNF) is a higher level of database normalization than the Third Normal Form (3NF). It addresses certain types of anomalies that can still occur in tables that are in 3NF. BCNF is named after Ronald Fagin, Raymond Boyce, and Edgar Codd, who contributed to its development in the 1970s.

A table is in Boyce-Codd Normal Form (BCNF) if, for every non-trivial functional dependency X -> Y (where X is a superkey), the entire X is a candidate key. In other words, in BCNF, all functional dependencies involve superkeys only, meaning that every determinant (X) is a candidate key.

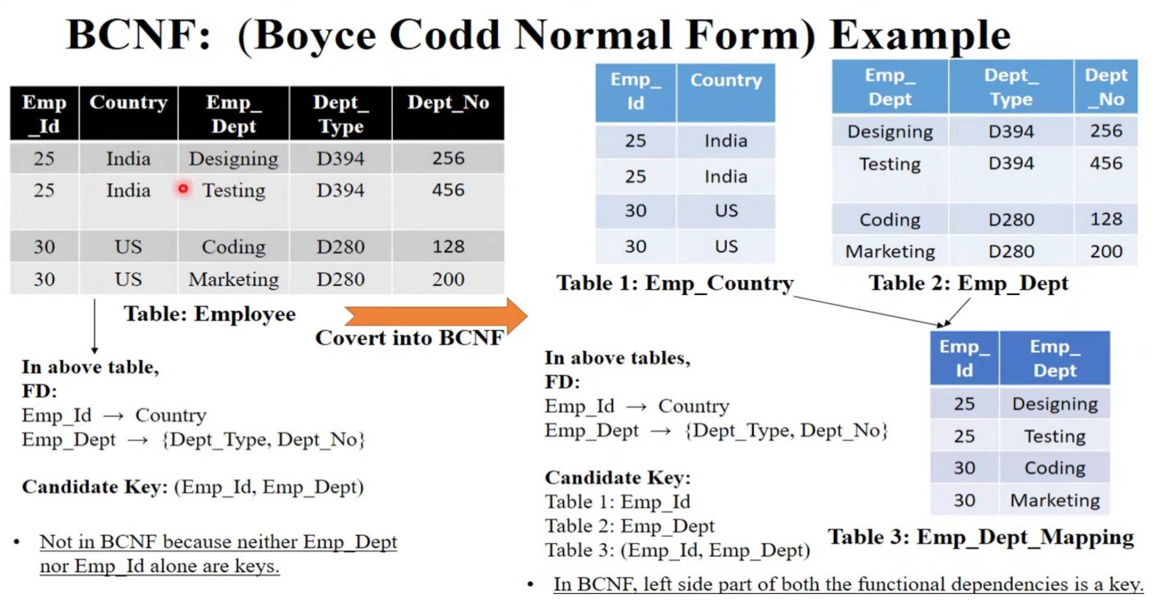
Key points of BCNF:
1. BCNF ensures that there are no partial dependencies. A partial dependency occurs when a non-key attribute depends on only part of the primary key.
2. BCNF eliminates anomalies related to non-prime attributes (attributes that are not part of any candidate key).
3. BCNF provides a higher level of data integrity and eliminates redundancy.

To better understand BCNF, let's consider an example:

Suppose we have a table called "Student_Courses" with the following attributes: {Student_ID, Course_Code, Course_Name, Instructor}. The functional dependencies in the table are as follows:

- Student_ID -> Course_Name (Each student takes only one course, so Student_ID is a candidate key).
- Course_Code -> Course_Name, Instructor (Each course has a unique code, so Course_Code is a candidate key).

The table is in 3NF because all non-key attributes depend on the candidate keys. However, it is not in BCNF because Course_Code is not a superkey, but it determines non-key attributes (Course_Name and Instructor).



BCNF: (Boyce Codd Normal Form) Example

Emp_Id	Country	Emp_Dept	Dept_Type	Dept_No
25	India	Designing	D394	256
25	India	Testing	D394	456
30	US	Coding	D280	128
30	US	Marketing	D280	200

Table: Employee

Covert into BCNF

In above table,
FD:
Emp_Id → Country
Emp_Dept → {Dept_Type, Dept_No}

Candidate Key: (Emp_Id, Emp_Dept)

- Not in BCNF because neither Emp_Dept nor Emp_Id alone are keys.

Emp_Id	Country
25	India
25	India
30	US
30	US

Table 1: Emp_Country

Emp_Dept	Dept_Type	Dept_No
Designing	D394	256
Testing	D394	456
Coding	D280	128
Marketing	D280	200

Table 2: Emp_Dept

Emp_Id	Emp_Dept
25	Designing
25	Testing
30	Coding
30	Marketing

Table 3: Emp_Dept_Mapping

In above tables,
FD:
Emp_Id → Country
Emp_Dept → {Dept_Type, Dept_No}

Candidate Key:
Table 1: Emp_Id
Table 2: Emp_Dept
Table 3: (Emp_Id, Emp_Dept)

- In BCNF, left side part of both the functional dependencies is a key.

Multivalued Dependencies and Fourth Normal Form

Multivalued Dependencies (MVDs) and Fourth Normal Form (4NF) are concepts in database normalization that address certain types of dependencies and anomalies that might still exist in tables that are in Boyce-Codd Normal Form (BCNF).

Multivalued Dependencies (MVDs):
Multivalued Dependencies represent a type of dependency in which one attribute's values determine multiple sets of values for another attribute, independent of the primary key. In simpler terms, an MVD occurs when an attribute has multiple independent sets of values associated with it for each value of another attribute. Multivalued Dependencies are denoted by the symbol "||".

Consider a table called "Student_Courses" with attributes {Student_ID, Course_Code, Course_Name, Instructors}, where Course_Code and Instructors are both multivalued attributes. The MVD in this table is:
- Student_ID ->> Course_Code, Instructors (Each student can take multiple courses with different instructors).

Fourth Normal Form (4NF):
Fourth Normal Form builds upon BCNF and addresses multivalued dependencies. A table is in Fourth Normal Form (4NF) if, for every non-trivial multivalued dependency X ->> Y, X is a superkey. In other words, 4NF ensures that all multivalued dependencies involve superkeys only.

To bring a table to 4NF, you need to decompose it further to eliminate multivalued dependencies while preserving the data's integrity.

Using the "Student_Courses" example, suppose we have the following data:

Student_ID	Course_Code	Course_Name	Instructors
101	MATH101	Math	ProfA, ProfB
102	SCI102	Science	ProfC

To bring the table to 4NF, we would decompose it into two separate tables:

Table: Students_Courses

Student_ID	Course_Code	Course_Name
101	MATH101	Math
102	SCI102	Science

Table: Courses_Instructors

Course_Code	Instructors
MATH101	ProfA
MATH101	ProfB
SCI102	ProfC

TRANSACTIONS: Transaction Concepts:

A transaction is a fundamental concept in database management systems (DBMS) and computer science. It represents a logical unit of work that consists of one or more operations performed on a database. The primary purpose of a transaction is to ensure that a sequence of related database operations are executed as a single, indivisible unit, providing four critical properties known as the ACID properties: Atomicity, Consistency, Isolation, and Durability.

Transaction Recovery:

Transaction recovery is a vital aspect of database management systems (DBMS) that ensures data consistency and integrity even in the face of failures or system crashes. It refers to the process of restoring the database to a consistent state after a transaction or system failure occurs. The primary goal of transaction recovery is to bring the database back to a state that reflects the effects of committed transactions while undoing the changes made by uncommitted or incomplete transactions.

Transaction recovery involves two essential mechanisms:

1. Undo (Rollback):
When a transaction fails or is aborted due to an error, the changes made by the transaction need to be undone to maintain data consistency. This process is known as undo or rollback. The DBMS rolls back the incomplete or uncommitted transaction, restoring the affected data to its original state before the transaction began.

2. Redo (Recovery):
After a system failure or crash, the committed transactions that have not yet been written to permanent storage need to be reapplied to the database to ensure durability. This process is known as redo or recovery. The DBMS applies the committed transactions that were lost during the crash, bringing the database back to its consistent state.

Key Concepts in Transaction Recovery:

1. Write-Ahead Logging (WAL): Write-ahead logging is a common technique used in transaction recovery. It involves writing the log records of all changes made by a transaction to a log file before the changes are actually applied to the database. This ensures that the log records are safely stored on disk before the corresponding data changes, allowing for proper recovery in case of failure.

2. Checkpoints: Checkpoints are periodic markers in the log file that indicate the state of the database at a specific point in time. They help expedite recovery by providing a known consistent state from which the redo and undo operations can start after a failure.

3. Crash Recovery: Crash recovery is the process of recovering the database after a system crash or failure. It involves applying the redo and undo operations to bring the database back to a consistent state.

4. Media Recovery: Media recovery is the process of recovering the database from a media failure, such as a disk failure. It involves restoring the data from backup and applying the redo6 and undo operations to bring the database up to date.

ACID Properties:

ACID is an acronym that stands for four fundamental properties that are critical for ensuring the reliability, consistency, and integrity of transactions in a database management system (DBMS). These properties provide a framework for maintaining data integrity and consistency even in the presence of failures, concurrency, and other unexpected events. The ACID properties are as follows:

1. Atomicity:
Atomicity ensures that a transaction is treated as an indivisible and all-or-nothing operation. It means that either all the operations within a transaction are executed successfully and the changes are committed to the database, or none of the operations are executed, and the database remains unchanged. If any part of the transaction fails or encounters an error, the entire transaction is rolled back, and the database is left in its original state. Atomicity prevents incomplete or partial transactions that could lead to data inconsistencies.

2. Consistency:
Consistency ensures that a transaction brings the database from one consistent state to another consistent state. In other words, a transaction should maintain the integrity and validity of the data, adhering to predefined rules and constraints. If the database is consistent before a transaction begins, it must remain consistent after the transaction is completed. If a transaction violates any integrity constraints or rules, it is aborted, and the database is rolled back to its original state.

3. Isolation:
Isolation ensures that each transaction is executed in isolation from other transactions running concurrently.Concurrent execution of multiple transactions can lead to interference, where one transaction's intermediate state is visible to other transactions. Isolation prevents interference between transactions by ensuring that the intermediate state of a transaction is not visible to other transactions until it is committed. It avoids race conditions and maintains data integrity and consistency.

4. Durability:
Durability guarantees that once a transaction is committed, its changes become permanent and are saved in the database, even in the event of a system failure or crash. Committed transactions survive system

failures and remain intact. The changes made by committed transactions are stored in non-volatile storage, ensuring data persistence and reliability.

Serializability:

Serializability is a concept in database management systems (DBMS) that ensures that the execution of multiple concurrent transactions produces the same result as if they were executed in a serial order, one after the other. It is a crucial property in multi-user database environments where multiple transactions may run simultaneously.

Types of serializability:

There are two main types of serializability in the context of database management systems: View Serializability and Conflict Serializability. Both types ensure that concurrent transactions produce the same result as some serial execution, but they use different approaches to achieve this goal.

1. View Serializability:
View Serializability is a weaker form of serializability compared to Conflict Serializability. It is based on the concept of "equivalence of serial schedules" and focuses on the final state of the database visible to users (the view) rather than the specific order of operations.

A schedule is considered view serializable if its result, when executed concurrently, is equivalent to the result of some serial execution of the same transactions. In other words, the final state of the database seen by all users after the concurrent execution is the same as the final state that would have been seen if the transactions were executed serially.

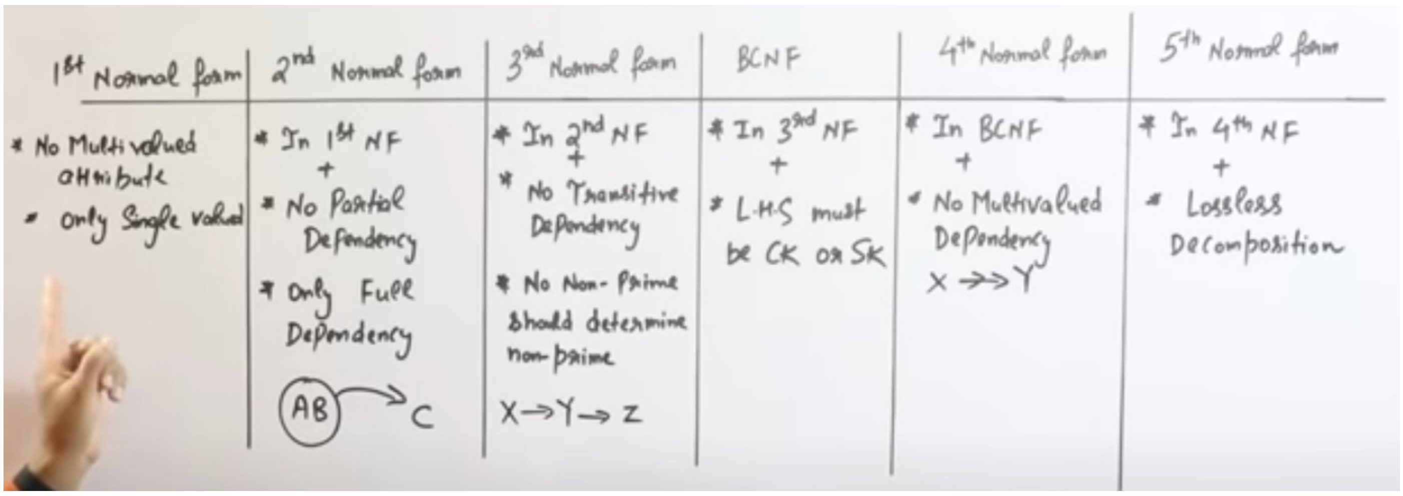
View Serializability allows for more flexible scheduling of transactions, but it may permit certain types of anomalies or interleavings that do not affect the final result.

2. Conflict Serializability:
Conflict Serializability is a stricter form of serializability compared to View Serializability. It focuses on the specific order of conflicting operations (read and write) performed by transactions.

A schedule is considered conflict serializable if it is equivalent to some serial schedule that preserves the order of all conflicting operations between transactions. Conflicting operations are operations that access the same data item and at least one of them is a write operation.

In other words, a schedule is conflict serializable if all the read and write operations on shared data items are correctly ordered in a way that avoids conflicts between transactions.

Conflict Serializability ensures that the final result of concurrent execution is the same as the result of some serial execution where all the conflicting operations are preserved in the same order.
Conflict Serializability is stricter than View Serializability and guarantees that the schedule avoids anomalies like dirty reads, non-repeatable reads, and write skew anomalies.



1st Normal form	2nd Normal form	3rd Normal form	BCNF	4th Normal form	5th Normal form
* No Multivalued attribute * only Single valued	* In 1st NF + * No Partial Dependency * Only Full Dependency $AB \rightarrow C$	* In 2nd NF + * No Transitive Dependency * No Non-Prime should determine non-prime $X \rightarrow Y \rightarrow Z$	* In 3rd NF + * L.H.S must be CK or SK	* In BCNF + * No Multivalued Dependency $X \twoheadrightarrow Y$	* In 4th NF + * Lossless Decomposition