coding75.com

DBMS Notes

(Database Management System)

Available here: coding75.com

Join coding75 Pro 🚀

Introduction to Database Management System

Before we dive into core concepts, let us see the basic terminologies:

What is Data?

Data encompasses raw, **unstructured information** such as text, observations, figures, symbols, and descriptions, lacking inherent purpose or significance. It is quantified in bits and bytes, fundamental units in computer storage and processing. Recorded data remains devoid of meaning until processed.

There are two types of data-

- **Quantitative** data
- **Qualitative** data

Quantitative data is expressed **numerically**, such as in measurements of weight, volume, or cost. On the other hand, **qualitative** data is **descriptive** rather than numerical, including attributes like names, gender, or hair color.

What is Information?

Information refers to **organized data** that provides **meaningful insights** or **knowledge**. For example, a weather forecast predicting rain tomorrow is information derived from data collected by meteorological instruments. Similarly, a report detailing quarterly sales figures provides information distilled from transaction records.

Difference between data and information

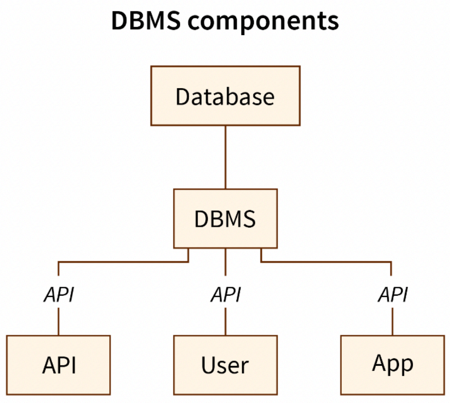
1. **Nature:** Data consists of raw, unprocessed facts, whereas information is data that has been processed, organized, and given context to make it meaningful and useful.
2. **Context:** Data lacks context and interpretation, while information is contextualized and interpreted to provide insights or understanding.
3. **Purpose:** Data alone does not serve a specific purpose, whereas information is intended to inform, guide decision-making, or provide understanding.
4. **Representation:** Data can be represented in various forms, such as text, numbers, or symbols, while information is typically presented in a structured and understandable format, such as reports, charts, or graphs.
5. **Actionability:** Data may not always be actionable on its own, while information is often actionable, providing guidance or prompting decisions based on the insights it conveys.

What is a database?

A **database** is a structured **collection** of **data** organized for efficient storage, retrieval, and manipulation. It typically consists of tables, each containing rows and columns, where each row represents a record and each column represents a field or attribute. Databases are managed by database management systems (DBMS) and are used in various applications to store and manage information, such as customer data, inventory records, or financial transactions.

Database Management System

A Database Management System (DBMS) is software designed to efficiently and securely manage databases. It provides an interface for users to interact with the database, allowing them to create, retrieve, update, and delete data. DBMS handles tasks such as data organization, storage, retrieval, indexing, security, and backup. Examples of popular DBMS include MySQL, Oracle Database, Microsoft SQL Server, and PostgreSQL.



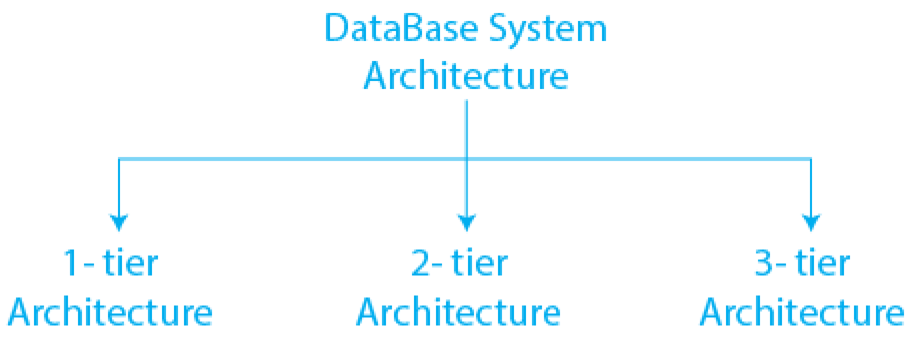
Difference between File System and DBMS

1. **Data Organization:** In a file system, data is typically stored in individual files with little to no organization beyond the file hierarchy. In contrast, a DBMS organizes data into structured formats like tables, rows, and columns within a database.
2. **Data Redundancy:** File systems often lead to redundant data storage, where the same information may be duplicated across multiple files. DBMS minimizes redundancy through normalization techniques, reducing storage requirements and ensuring data consistency.

3. **Data Integrity and Security:** DBMS offers better data integrity and security mechanisms compared to file systems. It enforces data integrity constraints, such as primary keys and foreign keys, and provides access control features to regulate who can access and modify data.
4. **Concurrency Control:** DBMS supports concurrent access by multiple users, providing mechanisms like locking and transactions to ensure data consistency and integrity during concurrent operations. File systems typically lack built-in concurrency control, leading to potential data corruption in multi-user environments.
5. **Data Retrieval and Manipulation:** DBMS provides powerful querying capabilities, allowing users to retrieve, filter, and manipulate data using structured query languages (e.g., SQL). In contrast, file systems offer limited querying capabilities, often requiring custom scripts or programs for data extraction and manipulation.

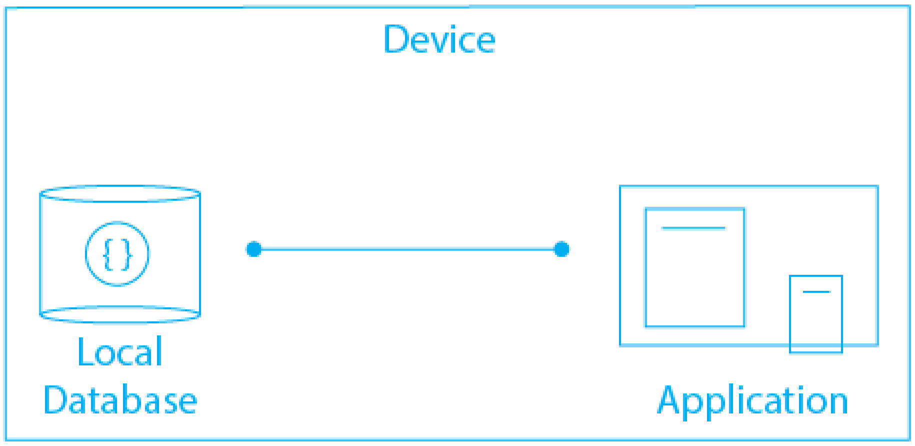
DBMS Application Architecture

There are 3 types of architecture for database management systems



1-Tier Architecture:

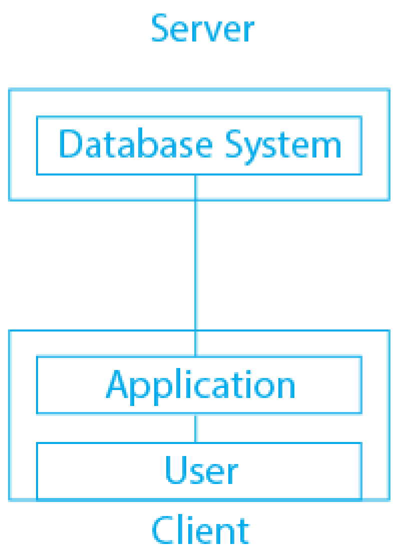
In a **1-Tier Architecture**, the database is accessible directly to the user. This means the user can directly access the Database Management System (DBMS) without any intermediary layers. In this setup, the client, server, and database all reside on the same machine. For instance, when learning SQL, one can set up an SQL server and database on their local system. This allows direct interaction with the relational database to execute operations. However, in practice, the industry typically opts for 2-tier or 3-tier architectures instead of this setup.



On a 1- tier applicatoin, the application and database are located on a same device

2-Tier Architecture:

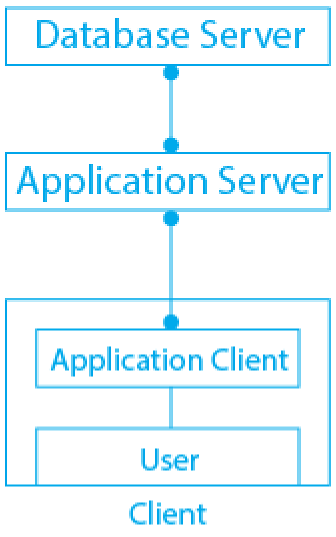
The **2-tier architecture** resembles a fundamental client-server model. The application on the client end communicates directly with the database located on the server side. APIs such as ODBC and JDBC facilitate this interaction. The server side handles tasks like query processing and transaction management. Meanwhile, the client side runs user interfaces and application programs. The client-side application establishes a connection with the server-side to interact with the DBMS.



In a 2- tier architecture, client connects directly to Database

3-Tier Architecture:

In **3-Tier Architecture**, an additional layer exists between the client and the server. Direct communication between the client and server is avoided. Instead, the client interacts with an application server, which in turn communicates with the database system. Query processing and transaction management occur at this server level. This intermediary layer serves as a conduit for the exchange of partially processed data between the server and the client. This architecture is commonly employed in large-scale web applications.



3- tier Architecture

View of Data (Three Schema Architecture)

In a **DBMS**, data visualization varies at different levels of **data abstraction**. Each level of abstraction allows developers to **shield users** from complex **data structures**. This is achieved by concealing intricate data structures behind layers of abstraction.

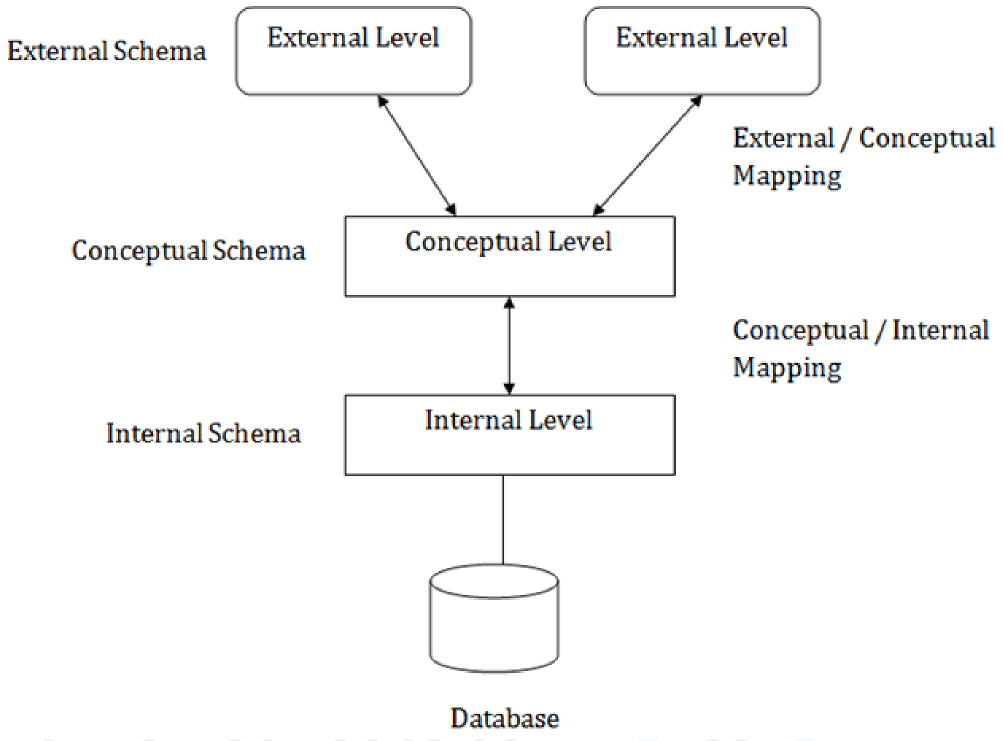
The primary goal of the three level architecture is to allow multiple users to access identical data with individualized perspectives, all while storing the core data only once.

Let’s see these 3 layers-

1. **Physical Level:**
    - Represents the lowest level of abstraction, detailing the storage mechanisms of the data.
    - Utilizes low-level data structures.

- Contains the Physical schema, delineating the physical storage arrangement of the database.
- Addresses aspects such as storage allocation methods (e.g., N-ary tree), data compression, encryption, and other storage optimizations.

2. **Logical/ Conceptual Level:**
    - Involves the conceptual schema, which outlines the database's design, including the stored data and their relationships.
    - Users operating at the logical level are not required to have knowledge of the underlying physical structures.
    - Database Administrators (DBAs), responsible for determining the database's contents, utilize the logical level of abstraction.
3. **View/ External Level:**
    - Represents the highest level of abstraction, aiming to simplify users' interaction with the system by offering different views to different end-users.
    - Each view schema describes the database portion relevant to a particular user group, concealing the remaining database content.
    - At the external level, a database consists of several schemas, sometimes referred to as subschemas, which delineate various views of the database.
    - Views also serve as a security mechanism, restricting user access to certain parts of the database.



What is Schema and Instances?

The **comprehensive design** of the **database** is referred to as the database **schema**. A schema **serves** as a **structural description** of data, typically remaining unchanged, while the data it describes may undergo frequent modifications.

Three types of schemas exist: **Physical, Logical**, and **View** schemas known as subschemas.

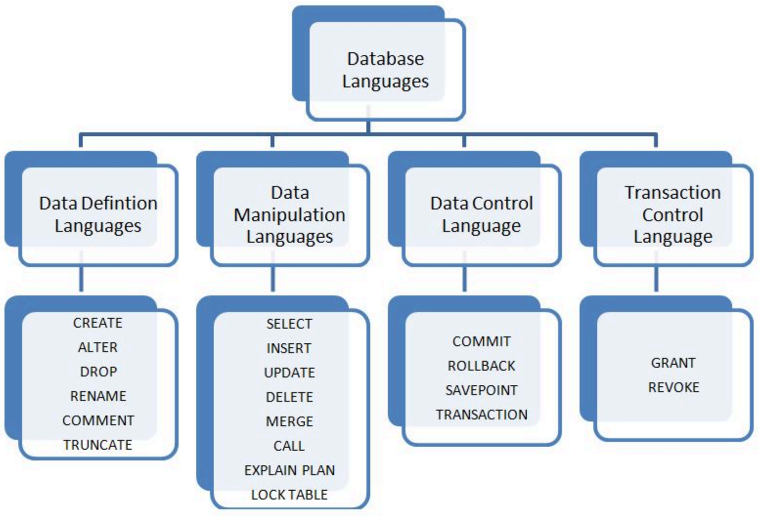
In DBMS, an **instance** refers to a **snapshot** or **representation** of the **database** at a **specific moment in time.** It includes all the data currently stored in the database, along with the current state of any database structures, such as indexes or constraints. An **instance** provides a **view** of the database as it exists at a particular point, capturing the data and structure at that moment.

Data Models

A **data model** is a conceptual framework for organizing and representing data in a database. It defines the structure, relationships, constraints, and operations associated with the data. There are several types of data models:

1. **Entity-Relationship Model:** Represents data as entities (objects), their attributes, and the relationships between entities. It is often used for designing databases before implementation in relational or other data models.
2. **Relational Data Model:** Represents data as tables with rows and columns, emphasizing relationships between tables. It is widely used in modern database systems, with SQL being the standard language for data manipulation.
3. **Object-Oriented Data Model:** Treats data as objects, similar to object-oriented programming languages. It supports inheritance, encapsulation, and polymorphism, providing more flexibility in representing complex data structures.

Data Languages

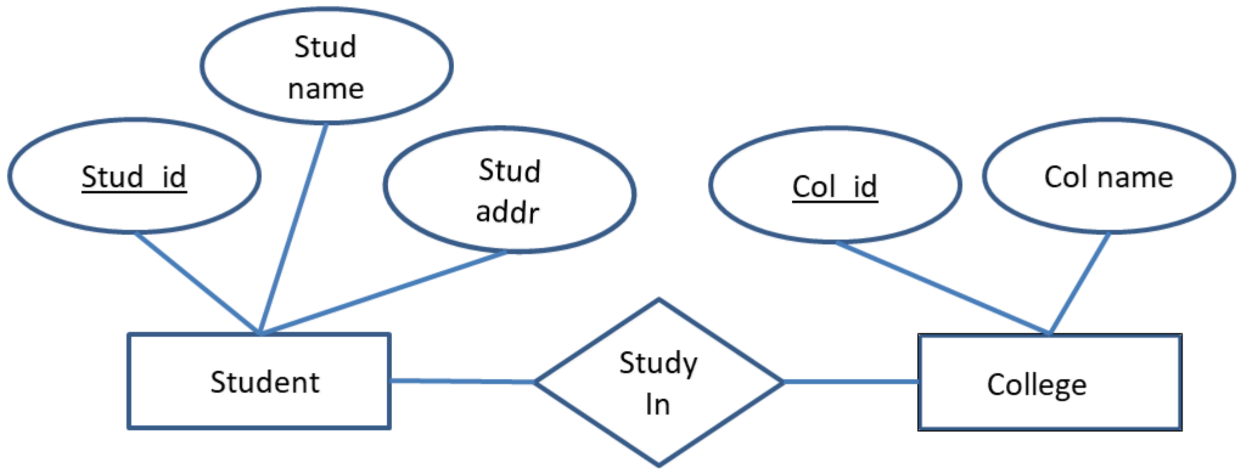


1. **Data Definition Language (DDL):** used to **define, modify**, and **manage** the **structure of database objects** such as **tables, views, indexes,** and **schemas**. **DDL commands** include CREATE, ALTER, DROP, and TRUNCATE, allowing users to create new database objects, modify existing ones, or remove them entirely.
2. **Data Manipulation Language (DML):** used to **manipulate data** within the database. **DML** commands include **SELECT, INSERT, UPDATE**, and **DELETE**, enabling users to retrieve, insert, modify, and delete data in database tables.
3. **Data Control Language (DCL):** used to **control access to data** within the database. **DCL** commands include **GRANT** and **REVOKE**, allowing users to grant or revoke permissions on database objects such as tables, views, and procedures.
4. **Transaction Control Language (TCL):** used to **manage transactions** within the database. **TCL** commands include **COMMIT, ROLLBACK**, and **SAVEPOINT**, enabling users to control the outcome and integrity of transactions.

Entity-Relationship diagram

An ER diagram, or Entity Relationship diagram, provides a visual depiction of the logical structure of a database.

- It illustrates the constraints and relationships between various components.
- An ER diagram typically consists of three primary components: Entity Sets, Attributes, and Relationship Sets.



Stud_id, Col_id is the primary key.

Student, College are entity sets of which (**Stud_id, Stud name, Stud addr**) and (**Col_id, Col name)** are attributes of respectively.

Study In is the relationship between **Student** and **College** entities.

Entity Set:

It is a set of the same type of entities.

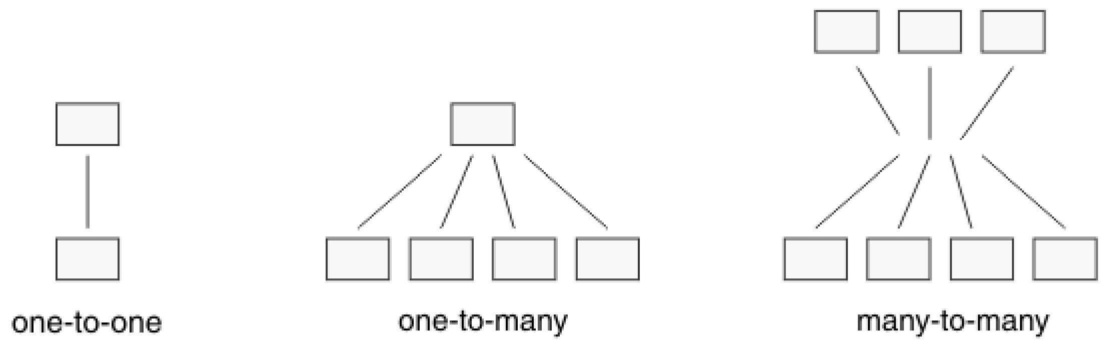
1. **Strong Entity Set:** in a strong entity set primary key exists. A primary key in an entity set is represented by underlining it.
2. **Weak Entity Set:** primary key does not exist. It contains a partial key called discriminator.

Relationship:

Association among entities. It can be unary, binary, ternary and n-ary.

Cardinality Constraint:

Maximum number of relationship instances an entity set can take part in. It can be one-to-one, one-to-many, many-to-one, many-to-many.



Attributes:

Attributes are the descriptive characteristics possessed by each entity within an Entity Set.

Types of attributes-

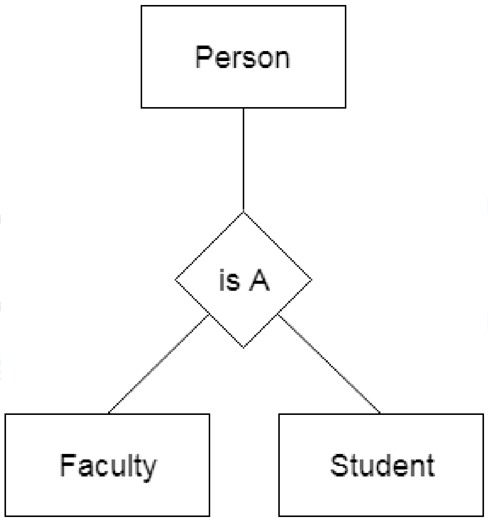
1. **Simple Attribute:** A single atomic value is associated with each instance of the attribute. For example, "Age" or "Name".
2. **Composite Attribute:** It comprises multiple simple attributes. For instance, an "Address" attribute may include "Street", "City", and "Zip Code" components.
3. **Derived Attribute:** This attribute's value is derived or calculated from other attributes. For example, "Age" can be derived from the "Date of Birth".
4. **Single-valued Attribute:** It holds a single value for each instance of the entity. For instance, "Height" or "Weight".
5. **Multi-valued Attribute:** It can hold multiple values for each instance of the entity. For example, "Phone Numbers" or "Skills".
6. **Key Attribute:** It uniquely identifies each entity within an entity set. For example, "ID" or "Employee Number".

Extended ER Features

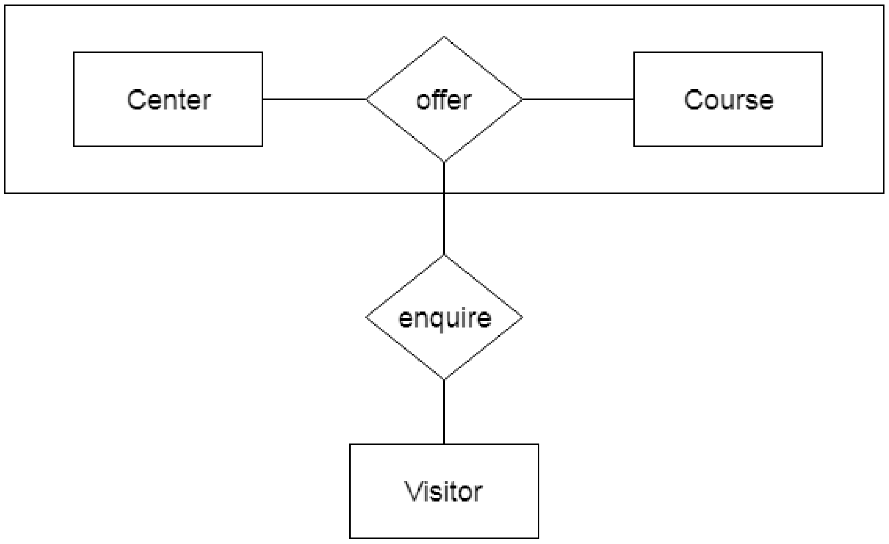
Extended Entity-Relationship (EER) features in database management systems (DBMS) extend the capabilities of traditional Entity-Relationship (ER) models.

These features include:

1. **Generalization:** Generalization aggregates common attributes and relationships from multiple entities to form a supertype.



2. **Specialization:** Specialization is the process of defining subtypes based on a subset of attributes and relationships from a supertype. In the above example if you move from top to bottom you can see specialization.
3. **Aggregation:** EER models allow entities and relationships to be aggregated into higher-level constructs known as aggregates. This enables the modeling of complex relationships between groups of entities or relationships. In the below diagram you can see center and course are being considered one entity, this helps to reduce the complexity of an ER-diagram and make relationships much clear.



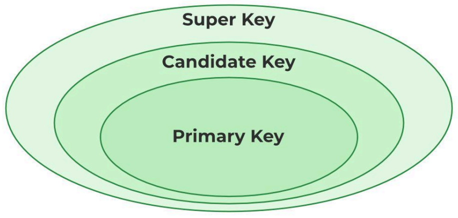
Keys in DBMS

A key is a set of attributes that can identify each tuple uniquely in the given relation. Types of keys:

1. **Primary Key:** A primary key uniquely identifies each record in a table. It must be unique and not null. There can be only one primary key per table.
2. **Candidate Key:** Candidate keys are attributes or a combination of attributes that could potentially serve as a primary key because they satisfy the uniqueness and non-null requirements. A table may have multiple candidate keys.
3. **Foreign Key:** A foreign key is a field in one table that refers to the primary key in another table. It establishes a relationship between two tables.
4. **Super Key:** A super key is a set of one or more attributes that, taken collectively, uniquely identify each record in a table. It may contain more attributes than necessary to uniquely identify records.
5. **Composite Key:** A composite key is a key composed of multiple attributes to uniquely identify records. Unlike primary keys, each

individual attribute in a composite key may not be unique on its own, but the combination of attributes is unique.

6. **Alternate Key:** Alternate keys are candidate keys that were not selected as the primary key. They could potentially serve as the primary key if the primary key is not available.



Functional dependency and types of functional dependency

Functional dependency is a concept in databases that describes the relationship between two attributes in a table. It means that the value of one attribute uniquely determines the value of another attribute. In simpler terms, if you know the value of one attribute, you can predict the value of another attribute based on this relationship.

Types of functional dependency:

1. **Trivial Functional Dependency:** Occurs when an attribute determines itself. For example, A → A.
2. **Non-Trivial Functional Dependency:** Occurs when an attribute determines another attribute that is not itself. For example, A → B, where B is not equal to A.

Decomposition of a relation

Decomposition of a relation refers to the process of splitting a single relation into two or more sub-relations.

Types of decomposition:

1. **Lossless Join Decomposition:** Ensures that no information is lost when decomposing a relation into smaller ones. The resulting sub-relations can be joined back together to obtain the original relation.
2. **Lossy Join Decomposition:** decomposition technique where joining the resulting sub-relations might lead to the loss of information, meaning that the original relation cannot be fully reconstructed.

Normalization

In DBMS, database normalization is the process of organizing and structuring the database to reduce redundancies and ensure data integrity through lossless decomposition.

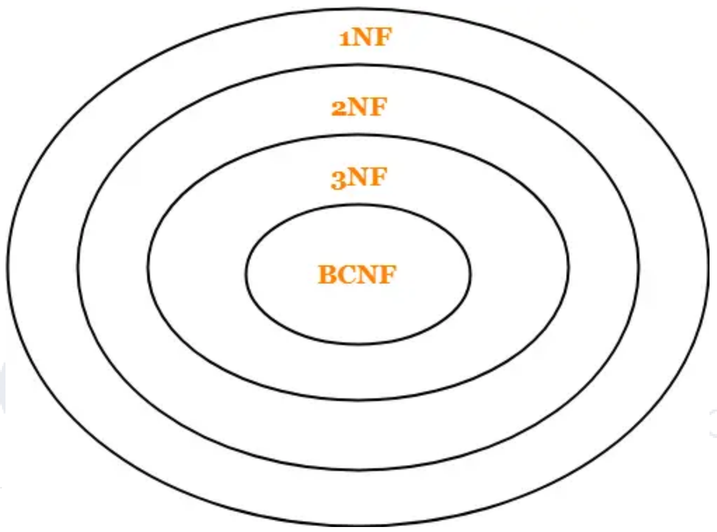
Normalization in DBMS is necessary to:

1. Reduce redundancy,
2. Ensure data integrity, and
3. Optimize storage and query performance.

Normal Forms

1. **First Normal Form (1NF):** Ensures that each table has atomic attributes, meaning each attribute contains only **indivisible values**. No repeating groups or arrays are allowed.
2. **Second Normal Form (2NF):** Requires that a table be in 1NF and that all non-key attributes are fully functionally dependent on the entire primary key, eliminating **partial dependencies**.
3. **Third Normal Form (3NF):** Requires a table to be in 2NF and ensures that there are no **transitive dependencies**, meaning that non-key attributes are not dependent on other non-key attributes.

4. **Boyce-Codd Normal Form (BCNF):** A given relation is considered to be in Boyce-Codd Normal Form (BCNF) if and only if it meets the following criteria:
    - The relation already exists in Third Normal Form (3NF).
    - For every non-trivial functional dependency 'A → B' in the relation, where 'A' and 'B' are sets of attributes, 'A' must be a superkey of the relation.



Transaction

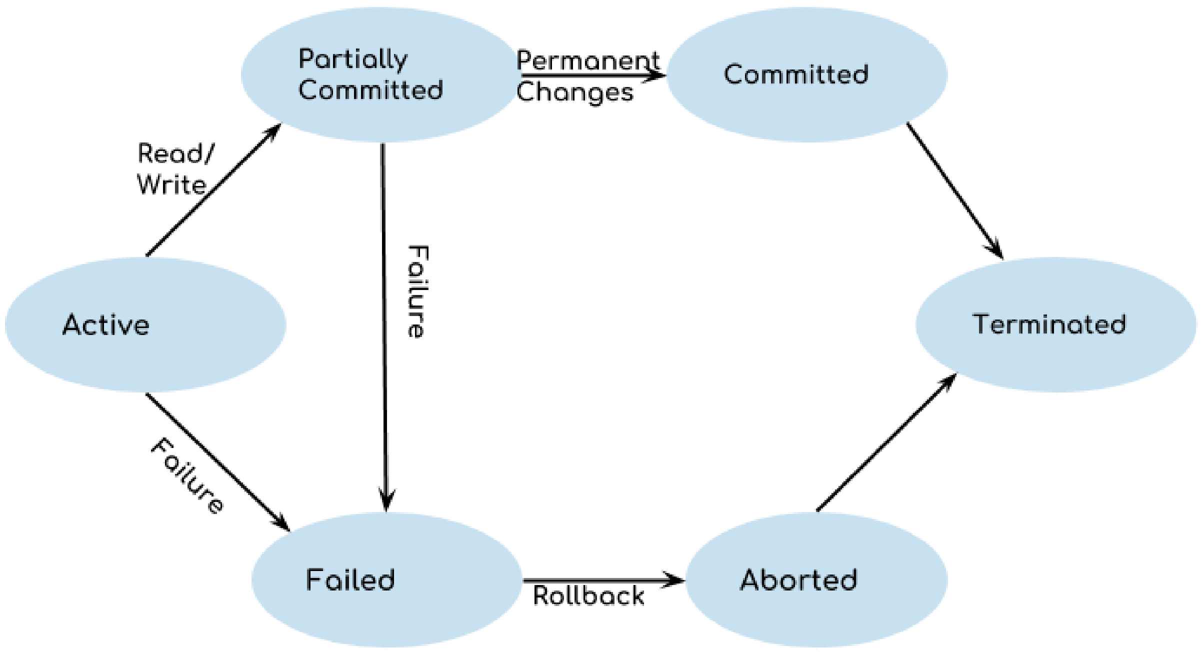
In a database management system (DBMS), a **transaction** represents a **unit of work** performed against the database. It is a sequence of one or more **database operations** (such as insert, update, delete, or select) that must be executed atomically, ensuring data integrity and consistency.

In transactions, two fundamental operations are commonly performed:

1. **Read Operation:** This operation involves reading the value of a specific data item, denoted as 'A', from the database. The retrieved value is then stored in a buffer located in the main memory.
2. **Write Operation:** This operation entails writing the updated value of the data item 'A' from the buffer back to the database. The buffer

contains the modified value, which is then written to the corresponding location in the database.

Transaction States



1. Active State:

This represents the **initial phase** in the **lifecycle** of a transaction. During the **active state**, the transaction's **instructions** are being **executed**. Any modifications made by the transaction are temporarily stored in the buffer located in the main memory.

2. Partially Committed State:

Once the **last instruction** of the **transaction** has been executed, it **transitions** into a **partially committed state**. In this state, the transaction is considered partially committed, as all changes made by the transaction are still stored in the buffer located in the main memory.

3. Committed State:

Once **all the changes** made by the **transaction** have been **successfully stored** in the **database**, it **transitions** into a **committed** state. At this stage, the transaction is considered fully committed, indicating that its modifications are now permanent and reflected in the database.

4. Failed State:

When a transaction is being executed in the active state or partially committed state and encounters a failure that prevents it from continuing execution, it transitions into a failed state.

5. Aborted State:

After the **transaction** has **failed** and entered a **failed state**, it becomes **necessary to undo** all the **changes** made by the **transaction**. To accomplish this, the transaction is **rolled back**, reverting any modifications it made to the database. Once the **rollback** process is **complete**, the transaction transitions into an **aborted state**.

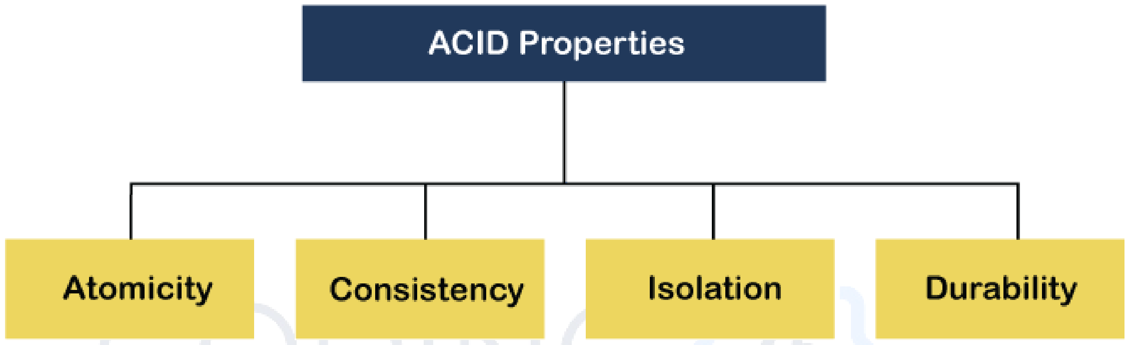
6. Terminated State:

This represents the **final stage** in the **lifecycle** of a **transaction**.

Once the **transaction** has **entered** either the **committed** state or the **aborted** state, it ultimately transitions into a terminated state, signifying the conclusion of its lifecycle.

ACID Properties

The properties that ensure the consistency, reliability, and correctness of transactions in a database system are known as the ACID properties. ACID stands for:



1. **Atomicity:**

   Ensures that a transaction is treated as a single unit of work, meaning that either all of its operations are successfully completed, or none of them are. If any part of the transaction fails, the entire transaction is rolled back, and the database returns to its original state.

2. **Consistency:**

   Ensures that the database remains in a consistent state before and after the execution of a transaction. Constraints and rules defined on the database schema must be enforced throughout the transaction, preserving the integrity of the data.

3. **Isolation:**

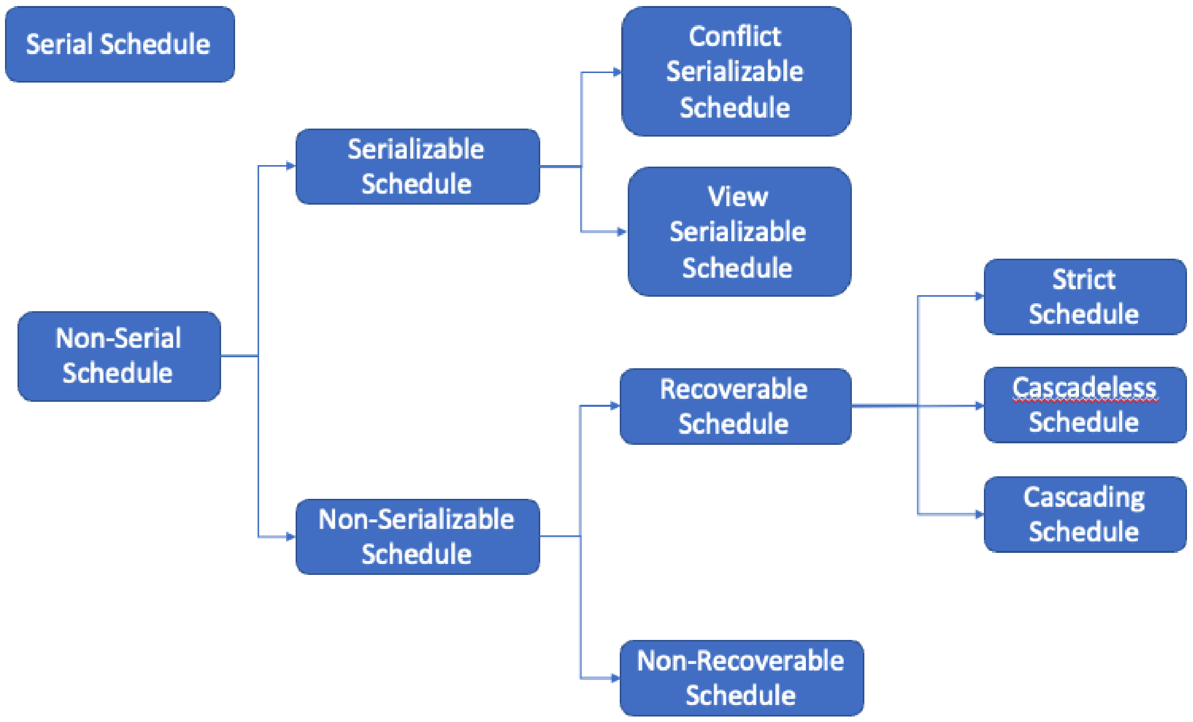
   Ensures that the execution of transactions is isolated from each other, preventing interference or corruption of data. Each transaction operates as if it were the only transaction executing against the database, even in the presence of concurrent transactions.

4. **Durability:**

   Ensures that the effects of a committed transaction are permanent and survive system failures. Once a transaction is committed, its changes are saved to the database and remain intact even in the event of a crash or power outage.

Schedules:

In a **database system**, a **schedule** refers to the **sequential order** in which the **operations** of **multiple transactions** are executed. It represents the timeline of **transactional operations** within the system.



1. Serial Schedule:

- **Transactions** execute one after the other, in a **sequential manner**.
- Only one transaction is allowed to execute at any given time.
- Serial schedules guarantee consistency, recoverability, cascadelessness, and strictness.

2. Non-Serial Schedule:

- Multiple transactions execute simultaneously.
- Operations of different transactions are interleaved or mixed with each other, allowing for parallel execution.
- Non-serial schedules may not always guarantee consistency, recoverability, cascadelessness, and strictness.

Serializability

Serializability is a concept used to determine the correctness of non-serial schedules and ensure database consistency. It identifies which concurrent schedules are valid and will preserve the integrity of the database.

1. **Serializable Schedules:**

   If a given non-serial schedule involving 'n' transactions is equivalent to some serial schedule of the same 'n' transactions, it is termed a serializable schedule. Serializable schedules consistently maintain the properties of consistency, recoverability, cascadelessness, and strictness.

   Types of serializable schedules:

   - **Conflict Serializable:** A non-serial schedule that can be transformed into a serial schedule by exchanging its non-conflicting operations is termed a conflict serializable schedule.
   - **View Serializable:** A schedule that, when compared to some serial schedule, is considered equivalent is termed a view serializable schedule.

2. **Non-Serializable Schedules:**

   A non-serial schedule that cannot be transformed into a serial schedule is referred to as a non-serializable schedule. Such a schedule is not guaranteed to yield the same outcome as any serial schedule on a consistent database.

Non-serializable schedules may or may not be consistent, may or may not be recoverable.

A. **Irrecoverable Schedules:** If a transaction reads a value that has been modified by another transaction that has not yet committed, and then commits before the transaction that made the modification, such a schedule is referred to as an Irrecoverable Schedule.
B. **Recoverable Schedules:** If a transaction performs a dirty read operation by accessing data modified by an uncommitted transaction, and its commit operation is delayed until the uncommitted transaction either commits or rolls back, then such a schedule is termed a Recoverable Schedule.

   Types of recoverable schedules:

   1) **Cascading Schedule:** If a schedule involves the failure of one transaction leading to the rollback or abort of several other dependent transactions, it is termed a Cascading Schedule, Cascading Rollback, or Cascading Abort.
   2) **Cascadeless Schedule:** If a schedule enforces that a transaction cannot read a data item until the last transaction that has written it is committed or aborted, it is termed a Cascadeless Schedule.
   3) **Strict Schedule:** If a schedule prohibits a transaction from both reading and writing a data item until the last transaction that has written it is committed or aborted, it is termed a Strict Schedule.

Thank You!

Join Coding75 Pro 🚀

- Live DSA Classes
- Project Building Session
- 1:1 Mock Interviews
- Resume Review
- CS Fundamental Classes
- And much more...

Join Now