Database - By Sourav Kapil (Data Scientist)

Intro to Databases

An introduction to databases typically covers the fundamental concepts and components of databases, including their purpose, structure, and management. Here's an overview to get started:

➢ What is a Database?

- A database is a structured collection of data organized in a way that facilitates efficient storage, retrieval, and manipulation of information.
- It serves as a central repository for storing and managing data that can be accessed by multiple users or applications.

➢ Purpose of Databases:

- Databases are used to store and manage large volumes of data systematically.
- They support data integrity, security, and concurrency control to ensure data consistency and reliability.
- Databases enable efficient data retrieval through queries, allowing users to extract specific information based on their needs.

➢ **Types of Databases:**

- **Relational Databases:** Organize data into tables with predefined relationships between them. Examples include MySQL, PostgreSQL, Oracle, and SQL Server.

- **NoSQL Databases:** Designed for handling unstructured or semi-structured data. Types include document-oriented (MongoDB), key-value stores (Redis), columnar (Cassandra), and graph databases (Neo4j).

Relational Databases vs Non-Relational Databases

Relational databases (RDBMS) and non-relational databases (NoSQL) are two types of database systems with distinct characteristics and use cases. Here's a comparison between them:

Aspect	**Relational Databases (RDBMS)**	**Non-Relational Databases (NoSQL)**
Data Structure	Organized into tables with predefined relationships (rows and columns) following a structured schema.	Use flexible data models like key-value pairs, documents, columnar, etc.
Scalability	Vertical scaling (adding resources to a single server). Horizontal scaling may be complex.	Horizontal scaling (adding more servers/nodes) without complexity. Better suited for handling large volumes of data and high loads.
Schema Flexibility	Fixed schema with predefined tables and relationships. Changes to schema can be challenging.	Schema flexibility allows dynamic changes without downtime. Agile for evolving data needs and rapid development.
Data Relationships	Emphasizes relationships using foreign keys, primary keys, and relational operations.	Focuses on denormalized data models and may not enforce strict relationships.

Query Language	Uses SQL (Structured Query Language). Supports complex queries, transactions, and joins.	Each type of NoSQL database may have its query language or API. Optimized for specific data models (e.g., document stores, key-value).
Use Cases	Well-suited for structured data, complex relationships, ACID transactions, relational reporting.	Excel in handling unstructured, semi-structured, or rapidly changing data (real-time analytics, content management, IoT data storage, etc.)

Choosing between relational and non-relational databases depends on factors like data structure, scalability needs, development agility, query complexity, and specific application requirements. Many modern applications leverage a combination of both types, known as polyglot persistence, to benefit from their respective strengths.

Database vs DBMS vs SQL

Database (DB):

- A database is an organized collection of structured data that is stored electronically in a computer system.
- It can be as simple as a single file containing data or a more complex system with multiple tables, relationships, and indexes.
- **Examples of databases i**nclude MySQL, PostgreSQL, MongoDB, Oracle, SQLite, and Microsoft SQL Server.

Database Management System (DBMS):

- A DBMS is software that allows users to create, manage, and interact with databases.
- It provides tools and functionalities for storing, retrieving, updating, and deleting data from databases.
- DBMS manages data integrity, security, concurrency control, and data access to ensure reliable and efficient data management.
- **Examples of DBMS** include MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, MongoDB, and SQLite.

Structured Query Language (SQL):

- SQL is a standardized language used for managing and querying relational databases.
- It provides commands and syntax for performing various operations on databases, such as creating and modifying database objects (tables, indexes, views), inserting, updating, and deleting data, and querying data using SELECT statements.
- SQL is divided into several categories, including Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL), and Transaction Control Language (TCL).
- **Examples of SQL commands** include CREATE TABLE, INSERT INTO, SELECT, UPDATE, DELETE, GRANT, REVOKE, COMMIT, and ROLLBACK etc.

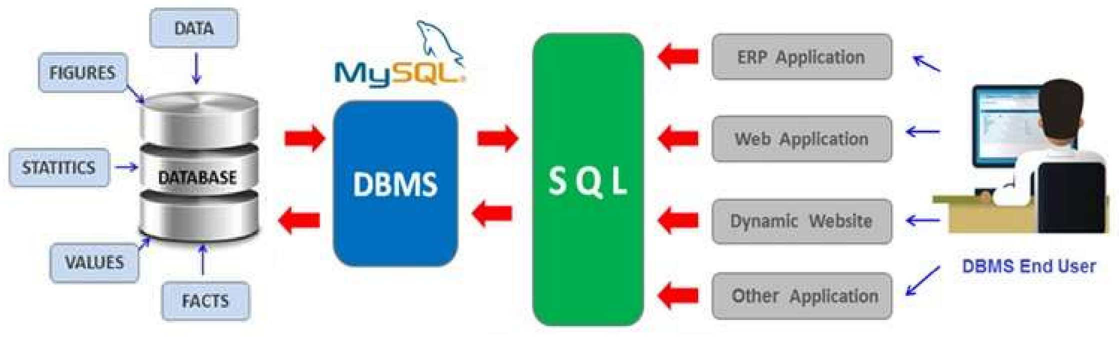
In summary:

➢ **Database (DB)** refers to the collection of structured data.

➢ **Database Management System (DBMS)** is the software that manages databases and provides tools for data management.

➢ **Structured Query Language (SQL)** is the language used to interact with relational databases and perform operations such as querying and manipulating data.

Together, these components form the foundation of modern data management systems, allowing users to create, manage, and query databases effectively.



Entity - Relationship (ER) Diagrams

An **Entity-Relationship (ER) diagram** is a visual representation of the entities (objects), attributes (properties), relationships, and constraints within a database. It's a powerful tool used in database design to model the structure and behaviour of data in a system. Here are the key components and concepts of ER diagrams:

1. Entities:

- An entity is a real-world object or concept with attributes that describe its characteristics.
- In an ER diagram, entities are represented by rectangles with the entity name inside.
- Examples of entities could be "Customer," "Product," "Order," etc.

2. Attributes:

- Attributes are the properties or characteristics of entities that provide detailed information about them.
- Each entity has attributes that describe its properties.
- Attributes are represented as ovals connected to their respective entities in the ER diagram.
- Examples of attributes for a "Customer" entity could be "CustomerID," "Name," "Email," etc.

3. Relationships:

- Relationships define how entities are related or connected to each other within the database.
- There are three types of relationships: one-to-one (1:1), one-to-many (1:N), and many-to-many (M:N).
- Relationships are represented by diamond shapes connecting related entities in the ER diagram.
- Examples of relationships could be "Customer places Order" (one-to-many) or "Product belongs to Category" (many-to-one).

4. Keys:

- Keys are attributes or combinations of attributes that uniquely identify each entity instance within a database.
- Primary keys uniquely identify each record in a table, while foreign keys establish relationships between tables.
- Primary keys are typically underlined in ER diagrams to denote their uniqueness and importance.

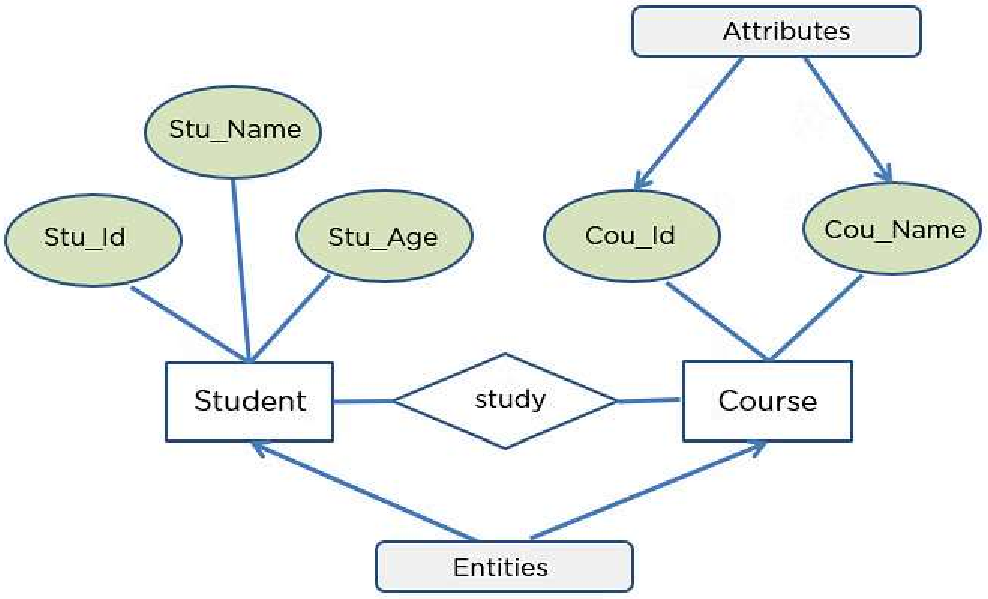
5. **Cardinality:**

- Cardinality represents the numerical relationship between entities in a relationship.
- For example, in a one-to-many relationship, the cardinality might be "one" on the side of the entity with one record and "many" on the side of the entity with multiple related records.

6. **Constraints:**

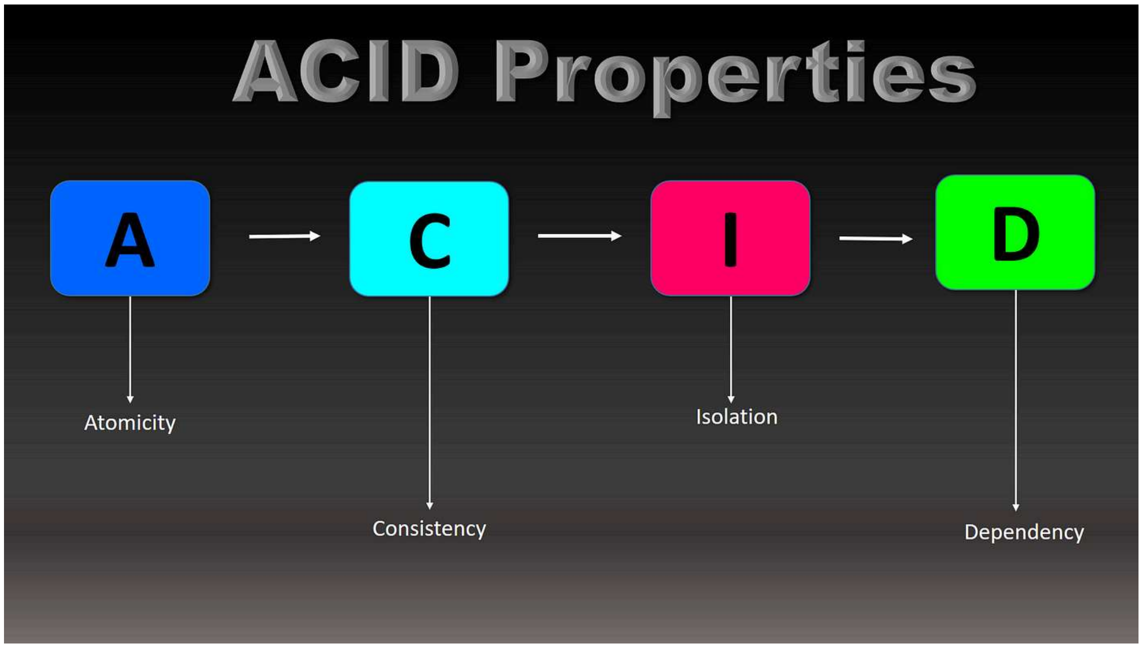
- Constraints define rules or conditions that must be satisfied within the database schema.
- Common constraints include entity integrity (ensuring each entity has a unique primary key), referential integrity (ensuring foreign key values match primary key values), and domain constraints (defining valid data ranges for attributes).

ER diagrams are essential for database designers, developers, and stakeholders to understand the data model, relationships, and constraints of a database system. They serve as a blueprint for database creation, modification, and maintenance throughout the development lifecycle.



ACID Properties

ACID (Atomicity, Consistency, Isolation, Durability) properties are fundamental principles in database management systems that ensure the reliability, consistency, and correctness of transactions. Here's a brief explanation of each of the ACID properties:



1. Atomicity:

- Atomicity ensures that each transaction is treated as a single, indivisible unit of work.
- Either all operations within the transaction are completed successfully and the changes are committed, or none of the operations are performed (rollback).
- This property guarantees that transactions are either fully completed or not executed at all, preventing partial updates that could lead to inconsistencies.

2. Consistency:

- Consistency ensures that the database remains in a valid state before and after each transaction.
- Transactions should preserve the integrity constraints, data types, and business rules defined in the database schema.
- If a transaction violates any constraint or rule, the entire transaction is rolled back, ensuring that the database is always consistent.

3. **Isolation:**

   - Isolation ensures that concurrent transactions do not interfere with each other's execution.
   - Each transaction should operate independently of other transactions, as if it were the only transaction running on the database.
   - Isolation prevents phenomena such as dirty reads, non-repeatable reads, and phantom reads by controlling the visibility of intermediate transaction states.

4. **Durability:**

   - Durability guarantees that once a transaction is committed, its changes are permanent and persist even in the event of system failures (e.g., power outage, hardware failure).
   - Committed transactions should be stored permanently in non-volatile storage (e.g., disk) and should survive crashes or restarts of the database system.
   - Durability is typically achieved through mechanisms like transaction logging and write-ahead logging (WAL), ensuring that data changes are durable and recoverable.

In summary, ACID properties provide a framework for ensuring the reliability, correctness, and integrity of transactions in database systems, which is essential for maintaining data consistency and preventing data corruption or loss. These properties are fundamental for ensuring the trustworthiness of modern database systems in various applications and industries.