Data: raw facts, figures, or *information* that can be processed or analyzed to gain insights or make decisions (numbers, text, images, videos, or any other measurable entity). data is typically *organized* in a structured format, such as tables or databases

Data Model: representation that defines the *structure, organization,* and constraints of data. It provides a framework for organizing and representing data in a consistent and logical manner. is used to specify how data is stored, accessed, and manipulated in a database system

- *Hierarchical Data Model*: Organizes data in a tree-like structure with parent-child relationships.
- *Network Data Model:* Similar to the hierarchical model but allows more complex many-to-many relationships.
- *Relational Data Model:* Uses tables (relations) to represent data and relationships between data.
- *Object-Oriented Data Model:* Organizes data using objects, similar to object-oriented programming.
- *Document Data Model:* Used in NoSQL databases to store data as documents (e.g., JSON, XML).

Data Structuring: *organizing data* in a way that enables efficient storage, retrieval, and manipulation. This process includes defining data formats, structures (like arrays, lists, trees), and schemas to ensure that data is stored in a consistent and accessible manner.

Database: *organized* collection of structured information or data that is *stored* electronically in a *computer* system. Databases are managed by Database Management Systems (DBMS) and are designed to efficiently store, retrieve, and manipulate data. They provide a systematic way to create, retrieve, update, and manage data in various formats, ensuring data integrity, security, and accessibility.

DBMS: a software application that interacts with users, applications, and the database itself to capture and analyze data. purpose of a DBMS is to provide a way to store, retrieve, and manipulate data in a database securely and efficiently (MySQL, PostgreSQL, Oracle Database, and MongoDB). It has:

- Data Retrieval: Allows users and applications to query and retrieve data.
- Data Manipulation: Provides tools for inserting, updating, and deleting data.
- Security and Access Control: Ensures data security and manages user permissions.
- Backup and Recovery: Provides mechanisms for data backup and recovery in case of failures.
- Data Storage: Efficiently stores data in databases.

SQL : (Structured Query Language) standardized programming *language* used to manage and manipulate *relational databases*. provides *commands* for creating, querying, updating, and deleting data within a database. SQL is known for its *declarative syntax* (specifies *what* the desired outcome is without explicitly listing the steps to achieve that outcome) and is widely used in applications requiring structured data storage and retrieval.

NoSQL : (Not Only SQL) a class of *database management systems* that do not rely on a fixed schema or use SQL as their primary query language. NoSQL databases are designed to handle large volumes of *unstructured* or semi-structured data and provide *flexible data models*, horizontal scalability, and high availability.

- Document Stores (e.g., MongoDB, CouchDB)
- Key-Value Stores (e.g., Redis, Amazon DynamoDB)
- Column-Family Stores (e.g., Apache Cassandra, HBase)
- Graph Databases (e.g., Neo4j, Amazon Neptune)

Keys and Relationships in SQL Databases keys and relationships are fundamental concepts for maintaining data integrity and establishing connections between tables:

Primary Key: a unique identifier for each record in a table. It ensures that each row in the table can be uniquely identified. A primary key must contain unique values and cannot contain NULL values.

Foreign Key: a column or set of columns in one table that refers to the primary key of another table. Foreign keys establish relationships between tables and enforce referential integrity.

Secondary Key (or Alternate Key): is a column or combination of columns that can be used to retrieve data but is not the primary key. Secondary keys are often used to create additional indexes to improve query performance.

Relationships: Relationships in SQL databases define how tables relate to each other. The main types of relationships are:

- *One-to-One:* Each record in one table corresponds to exactly one record in another table.
- *One-to-Many:* A record in one table corresponds to multiple records in another table.
- *Many-to-Many*: Multiple records in one table correspond to multiple records in another table, often managed through a junction table.

Data Types in SQL Databases : a variety of data types to store different kinds of data. Common data types include:

- Integer Types: `INT`, `SMALLINT`, `BIGINT`
- Floating-Point Types: `FLOAT`, `DOUBLE`, `DECIMAL`
- String Types: `CHAR`, `VARCHAR`, `TEXT`
- Date and Time Types: `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`
- Boolean Type: `BOOLEAN`
- Binary Types: `BLOB`, `BINARY`

INT

Storage: Typically requires 4 bytes (32 bits) of storage.

Range:

Signed: From -2,147,483,648 to 2,147,483,647
Unsigned: From 0 to 4,294,967,295

Use Case: Suitable for general-purpose integer storage, such as ID columns or quantities, where the values are expected to be within the typical range of an integer.

SMALLINT

Storage: Typically requires 2 bytes (16 bits) of storage.

Range:

Signed: From -32,768 to 32,767
Unsigned: From 0 to 65,535

Use Case: Ideal for cases where the range of values is known to be small, such as age, rating, or small counts. This type can save storage space and improve performance when large numbers of rows are involved.

BIGINT

Storage: Typically requires 8 bytes (64 bits) of storage.

Range:

Signed: From -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
Unsigned: From 0 to 18,446,744,073,709,551,615.

Use Case: Suitable for applications requiring very large integer values, such as tracking very large counts, unique identifiers in distributed systems, or representing timestamps in microseconds.

FLOAT	**Storage**: Typically uses 4 bytes (32 bits), though this can vary by database system. **Precision**: `FLOAT` has a limited precision, which can lead to rounding errors. It is usually good for about 7 decimal digits of precision. **Use Case**: Suitable for scientific calculations where an exact value isn't critical, and the range of values can be very large or very small. Examples include storing sensor data or scientific measurements. **Syntax**: In some databases, you can specify the precision and scale with `FLOAT(M, D)`, where `M` is the number of digits and `D` is the number of decimal places.
DOUBLE	**Storage**: Typically uses 8 bytes (64 bits). **Precision**: `DOUBLE` provides about 15-16 decimal digits of precision, making it more precise than `FLOAT`. **Use Case**: Used in applications where a higher precision is necessary compared to `FLOAT`, such as financial calculations that require accuracy over large ranges. **Syntax**: In some systems, specified as `DOUBLE(M, D)` for precision control, but often the full precision is used by default.
DECIMAL	**Storage**: The storage size depends on the precision specified. It can be variable but is usually more than `FLOAT` or `DOUBLE` for high precision. **Precision**: Allows exact representation of decimal numbers, as it stores numbers as strings of digits, providing precise arithmetic without rounding errors typically associated with floating-point types. **Use Case**: Ideal for financial data and other applications where precision is critical, such as currency calculations, where rounding errors can lead to significant discrepancies. **Syntax**: Specified as `DECIMAL(M, D)`, where `M` is the maximum number of digits (precision) and `D` is the number of digits to the right of the decimal point (scale).
CHAR	**Storage**: Requires a fixed amount of storage, based on the defined length, regardless of the actual string length. **Performance**: Generally provides faster access compared to `VARCHAR` for small, fixed-size strings, as it has consistent storage space. **Use Case**: Ideal for storing data that has a consistent length, such as country codes (e.g., "USA"), fixed-length product codes, or other uniformly sized data. **Syntax**: `CHAR(n)`, where `n` is the length of the string.

VARCHAR	**Storage**: More storage-efficient than `CHAR` for strings that vary significantly in length. It uses less space for shorter strings but may require more overhead for length storage. **Performance**: Slightly less efficient than `CHAR` for short, consistently sized data due to the overhead of variable-length management, but more efficient for variable-length data. **Use Case**: Suitable for columns where the length of the data can vary significantly, such as names, email addresses, or descriptions. **Syntax**: `VARCHAR(n)`, where `n` is the maximum length of the string.
TEXT	**Storage**: Typically allows for very large sizes (depending on the database system, this can be up to 65,535 characters or more). **Performance**: `TEXT` fields can be slower to access and manipulate than `CHAR` or `VARCHAR` fields, especially for very large text, due to increased complexity and storage requirements. **Use Case**: Suitable for storing large blocks of text such as product descriptions, comments, or articles. **Syntax**: Simply defined as `TEXT` without a specified length.
BLOB	(Binary Large Object) is typically used to store unstructured data (images, audio, video, and other multimedia files). **Storage Size**: `BLOB` fields can store a significant amount of data, ranging from kilobytes to gigabytes, depending on the database system. **Types of BLOBs**: Some database systems differentiate between types of BLOBs based on the size of the data they can store: **TINYBLOB**: Stores up to 255 bytes of data. **BLOB**: Stores up to 65,535 bytes (64 KB). **MEDIUMBLOB**: Stores up to 16,777,215 bytes (16 MB). **LONGBLOB**: Stores up to 4,294,967,295 bytes (4 GB). **Use Cases**: Ideal for storing multimedia files such as images, audio, and video. Used when the data does not have a fixed size or structure and requires storage in raw binary format.

BINARY	Unlike `BLOB`, `BINARY` is meant for smaller, fixed-size binary data rather than large objects. **Fixed Length**: Similar to `CHAR`, `BINARY` stores data with a fixed length, padding with zeros if the data is shorter than the specified length. **Storage Size**: The storage size is determined by the defined length of the `BINARY` field. **Use Cases**: Suitable for storing small, fixed-size binary data such as encryption keys, hash values, or binary flags. Useful when data length is consistent and relatively small.

Declarative Language : type of programming language where the programmer specifies *what* the program should accomplish, rather than explicitly defining *how* to achieve that result. focus on describing the desired outcome without detailing the control flow or algorithmic steps needed to achieve it. SQL is an example of a declarative language

Standardized Language: a programming language or protocol that has been formally approved by a recognized standards organization, such as ISO (International Organization for Standardization) or ANSI (American National Standards Institute). ensures consistency, interoperability, and compatibility across different systems and platforms. SQL, for example, is a standardized language with defined syntax and semantics that allow it to be used consistently across various relational database management systems (RDBMS).