RDBMS (Relational Database Management System)

- **Definition**: An RDBMS is a type of database management system (DBMS) that stores data in structured tables (relations) and enforces relationships between those tables using keys (primary and foreign keys).
- **Key Features**:
  - **Tables**: Data is organized into tables (relations), each with rows (records) and columns (attributes).
  - **Schemas**: Defines the structure of the database, including tables, fields, data types, and relationships.
  - **SQL**: Structured Query Language is used to query, manipulate, and manage data.
  - **ACID Compliance**: RDBMS often adheres to the ACID properties to ensure data integrity.
- **Popular Examples**: MySQL, PostgreSQL, Oracle Database, SQL Server.

---

2. ACID Properties

ACID stands for the four key properties that guarantee the reliable processing of database transactions:

- **Atomicity**: A transaction is all-or-nothing; it either completes entirely or does not execute at all. If any part of the transaction fails, the entire transaction is rolled back.
- **Consistency**: The database must remain in a valid state before and after the transaction. Any transaction must move the database from one valid state to another.
- **Isolation**: Transactions are isolated from one another. The intermediate states of a transaction are invisible to other transactions, ensuring data consistency.
- **Durability**: Once a transaction is committed, it is permanent and will survive system failures, ensuring the data is saved to persistent storage.

---

3. Dirty Read, Repeatable Read, Non-repeatable Read, Phantom Reads

These terms refer to different types of **transaction anomalies** that can occur when multiple transactions interact with the same data concurrently. They are related to the **isolation level** of transactions in a database.

- **Dirty Read**:
  - A **dirty read** occurs when a transaction reads data that has been modified by another uncommitted transaction. If the second transaction rolls back, the first transaction will have read invalid data.
  - **Example**: Transaction A updates a record but doesn't commit. Transaction B reads this record before A commits or rolls back. If A rolls back, B has read data that never existed.
- **Repeatable Read**:

    - In a **repeatable read**, if a transaction reads a piece of data, it will always see the same value (even if other transactions modify the data). This prevents **non-repeatable reads**, but **phantom reads** can still occur.
    - **Example**: Transaction A reads a value, and no other transaction can change that value until A finishes.
- **Non-repeatable Read**:
    - A **non-repeatable read** occurs when a transaction reads the same piece of data twice and gets different values because another transaction modified the data between the two reads.
    - **Example**: Transaction A reads a record, and Transaction B updates it before A reads the same record again.
- **Phantom Reads**:
    - **Phantom reads** occur when a transaction reads a set of rows based on a query condition (e.g., a range of values), but another transaction inserts or deletes rows that match the condition, causing the result set to change unexpectedly.
    - **Example**: Transaction A selects all customers with a balance greater than $1000. Meanwhile, Transaction B inserts a new customer with a balance of $1200. When A re-executes the query, the new customer appears in the result set.

---

4. NoSQL (Not Only SQL)

- **Definition**: NoSQL refers to a broad class of database management systems that do not use the relational model or SQL as their primary interface. These databases are designed to handle large volumes of unstructured or semi-structured data and provide flexibility in terms of schema and scalability.
- **Types of NoSQL Databases**:
    - **Document-oriented**: Store data as documents (e.g., MongoDB, CouchDB).
    - **Key-Value Stores**: Store data as key-value pairs (e.g., Redis, DynamoDB).
    - **Column-family Stores**: Store data in columns rather than rows (e.g., Cassandra, HBase).
    - **Graph Databases**: Store data as nodes and edges (e.g., Neo4j).
- **Key Features**:
    - **Scalability**: NoSQL databases often scale horizontally by adding more machines.
    - **Schema flexibility**: No predefined schema for the data, allowing dynamic changes.
    - **CAP Theorem**: Focus on different trade-offs between Consistency, Availability, and Partition tolerance.

---

5. Normalization

- **Definition**: Normalization is the process of organizing data in a database to minimize redundancy and avoid undesirable characteristics like insertion, update, and deletion anomalies.
- **Normal Forms**:

    - **First Normal Form (1NF)**: Eliminate duplicate columns from the same table. Ensure each column contains atomic values (no sets or arrays).
    - **Second Normal Form (2NF)**: Ensure the table is in 1NF and that all non-key columns are fully dependent on the primary key (eliminate partial dependency).
    - **Third Normal Form (3NF)**: Ensure the table is in 2NF and that all non-key columns are directly dependent on the primary key, not on other non-key columns (eliminate transitive dependency).
    - **Boyce-Codd Normal Form (BCNF)**: A stricter version of 3NF.
    - Higher normal forms (4NF, 5NF) deal with more complex cases of multivalued dependencies and join dependencies.
- **Purpose**: The goal of normalization is to organize the data to reduce redundancy and improve data integrity.

---

6. Indexing

- **Definition**: Indexing is a technique used to speed up the retrieval of data from a database by creating a data structure (usually a B-tree or hash table) that allows quick access to rows in a table based on the values of one or more columns.
- **Types of Indexes**:
    - **Single-column index**: Indexes a single column in the table.
    - **Composite (multi-column) index**: Indexes multiple columns together.
    - **Unique index**: Ensures that all values in the indexed column(s) are unique.
    - **Full-text index**: Used for indexing text fields to support full-text search operations.
    - **Clustered index**: Alters the physical order of the data in the table based on the indexed column.
    - **Non-clustered index**: A separate structure that holds a reference to the rows in the table.
- **Benefits of Indexing**:
    - **Faster queries**: Indexes significantly speed up retrieval operations, especially for large tables.
    - **Efficient sorting and searching**: Helps to quickly find rows that match certain criteria and support ordering.
- **Drawbacks**:
    - **Slower insertions, updates, and deletions**: Indexes need to be updated whenever the data in the table changes.

- **Database Backup and Recovery**
- **1. Backup:**
- A database backup is the process of creating a copy of the database (or parts of it) to protect against data loss due to corruption, hardware failure, or other issues.
- **Types of Backups**:
- **Full Backup**: A complete copy of the entire database.
- **Incremental Backup**: Backs up only the changes made since the last backup (either full or incremental).
- **Differential Backup**: Backs up changes made since the last full backup.
- **Transaction Log Backup**: Backs up the transaction log, which records all changes made to the database.
- **Backup Strategies**:
- **Cold Backup**: Backing up the database while it's offline (no users can access it).
- **Hot Backup**: Backing up the database while it is running (online, available to users).
- **Automated Backup**: Scheduling backups to occur at regular intervals.
- **Backup Tools**: Different databases have different tools:
- For MySQL: mysqldump
- For PostgreSQL: pg_dump
- For SQL Server: Backup command or SQL Server Management Studio (SSMS)
- **2. Recovery:**
- Database recovery refers to restoring a database from a backup to recover from data loss, corruption, or failure.
- **Types of Recovery**:
- **Full Recovery**: Restoring from the last full backup and any transaction logs to bring the database to a specific point in time.
- **Point-in-Time Recovery (PITR)**: Restoring the database to a specific point in time by applying transaction logs after a full backup.
- **Crash Recovery**: Automatically recovering a database after an unexpected shutdown or failure by using transaction logs.
- **Recovery Steps**:
- **Restore Full Backup**: Restore the last full backup.
- **Apply Differential Backup** (if applicable).
- **Apply Transaction Logs**: Apply any transaction logs to roll forward changes.
- **Recovery Models**:
- **Simple Recovery Model**: Only full and differential backups are supported. No transaction logs.
- **Full Recovery Model**: Full backup and transaction log backups, allowing point-in-time recovery.
- **Bulk-Logged Recovery Model**: Similar to full, but optimized for bulk operations.
- ---
- **SQL Operations: SELECT, INSERT, UPDATE, DELETE**
- These are the core SQL commands for manipulating and retrieving data in a relational database.
- **1. SELECT**
- The SELECT statement is used to retrieve data from a database.

- **Basic Syntax**:
- sql
- Copy
- SELECT column1, column2 FROM table_name;
- You can specify specific columns, or use * to select all columns.
- sql
- Copy
- SELECT * FROM employees;
- **Filtering Results**: You can filter the results using WHERE to specify conditions.
- sql
- Copy
- SELECT * FROM employees WHERE age > 30;
- **Sorting Results**: Use ORDER BY to sort the results.
- sql
- Copy
- SELECT * FROM employees ORDER BY name ASC;
- **Limit Results**: Use LIMIT to restrict the number of rows returned.
- sql
- Copy
- SELECT * FROM employees LIMIT 10;
- **Join Tables**: Combine rows from multiple tables using JOIN.
- sql
- Copy
- SELECT employees.name, departments.name
- FROM employees
- JOIN departments ON employees.department_id = departments.id;
- **2. INSERT**
- The INSERT statement is used to add new records to a table.
- **Basic Syntax**:
- sql
- Copy
- INSERT INTO table_name (column1, column2)
- VALUES (value1, value2);
- **Inserting Multiple Rows**:
- sql
- Copy
- INSERT INTO employees (name, age, department)
- VALUES ('John Doe', 28, 'HR'),
- ('Jane Smith', 34, 'IT');
- **Insert Without Specifying Columns** (if all columns are being filled):
- sql
- Copy
- INSERT INTO employees
- VALUES (1, 'Alice', 30, 'Finance');
- **3. UPDATE**
- The UPDATE statement is used to modify existing records in a table.
- **Basic Syntax**:
- sql
- Copy

- UPDATE table_name
- SET column1 = value1, column2 = value2
- WHERE condition;
- **Example**:
- sql
- Copy
- UPDATE employees
- SET age = 29
- WHERE name = 'John Doe';
- **Important**: Always use the WHERE clause with UPDATE to avoid updating all rows in the table. Without it, the update affects every row in the table.
- **4. DELETE**
- The DELETE statement is used to remove records from a table.
- **Basic Syntax**:
- sql
- .
- DELETE FROM table_name
- WHERE condition;
- **Example**:
- sql
- .
- DELETE FROM employees
- WHERE name = 'John Doe';
- **Caution**: Like UPDATE, be careful when using DELETE without a WHERE clause. If omitted, it will delete all rows in the table.
-

-
- **Disk space**: Indexes require additional storage.