Khaleeda

Database

A **Database Management System (DBMS)** is software that allows users to define, create, maintain, and interact with databases. It provides an interface for storing, retrieving, and manipulating data efficiently while ensuring data security, integrity, and consistency.

A **Relational Database Management System (RDBMS)** is a type of database management system that organizes data into structured tables (also called relations), where each table consists of rows and columns.

Advantages of RDBMS:

1. Structured and Organized Data

Data is stored in tables with rows and columns, providing a clear and logical structure.

Relationships between data are maintained through primary and foreign keys.

2. Data Integrity

Ensures data accuracy and consistency using constraints like primary keys, foreign keys, unique constraints, and check constraints.

Helps maintain relationships and prevent data anomalies.

3. Data Security

Provides robust access control mechanisms, such as user authentication and permissions, to restrict access to sensitive data.

Supports data encryption and auditing features.

4. Scalability

Can handle increasing amounts of data efficiently, particularly with optimization techniques like indexing and partitioning.

5. Standardized Query Language

Uses SQL (Structured Query Language) for data querying, manipulation, and management, making it accessible and widely supported.

Standardized SQL ensures compatibility across different RDBMS systems.

6. Concurrency Control

Supports multiple users simultaneously while maintaining data consistency through transaction management and locking mechanisms.

Ensures that concurrent operations do not lead to data conflicts.

7. Backup and Recovery

Provides built-in tools for automated backup and recovery to safeguard against data loss.

Ensures business continuity by enabling point-in-time recovery.

8. Data Redundancy Elimination

Reduces data duplication through normalization, saving storage space and maintaining data integrity.

Tables:

Store data in a structured format using rows (records) and columns (attributes).

Example: A Users table with attributes like UserID, Name, Email.

Schemas:

Define the logical structure of the database, including tables, views, constraints, and relationships.

Acts as a blueprint for how data is organized.

Keys

Primary Key: Unique identifier for each record in a table (e.g., UserID in a Users table).

Foreign Key: Establishes a relationship between two tables by referencing a primary key in another table.

Candidate Key: Potential keys that can serve as the primary key.

Composite Key: A key made up of two or more columns.

Relationships:

Define how data in different tables is connected (e.g., One-to-One, One-to-Many, Many-to-Many).

Normalization:

Process of organizing data to eliminate redundancy and ensure data dependencies are logical.

Involves normal forms (1NF, 2NF, 3NF, etc.).

Transactions:

A group of database operations treated as a single unit to ensure data integrity.

Properties: ACID (Atomicity, Consistency, Isolation, Durability).

Constraints:

Rules enforced on data in tables to maintain accuracy and integrity.

Types: NOT NULL, UNIQUE, CHECK, DEFAULT, PRIMARY KEY, FOREIGN KEY.

Indexes:

Improve data retrieval speed by creating a quick lookup mechanism for table rows.

Views:

Virtual tables based on the result of a SQL query. They provide a simplified or specific perspective of the data.

Triggers:

Automatic actions executed in response to specific events on a table (e.g., insert, update, delete).

Stored Procedures:

Predefined SQL queries and logic stored in the database to perform specific tasks.

Backup and Recovery:

Mechanisms to save database data and restore it in case of failures.

Data Integrity:

Ensures accuracy, consistency, and reliability of data through constraints and validation.

Query Language:

SQL (Structured Query Language) is used to query, insert, update, and manage data.

Relationships Between Tables:

One-to-One: Each record in Table A has one related record in Table B.

One-to-Many: One record in Table A can relate to multiple records in Table B.

Many-to-Many: Records in Table A relate to multiple records in Table B and vice versa.

DDL (Data Definition Language)

- **Purpose:** Used to define and manage the structure of a database (e.g., creating, altering, and deleting tables and other database objects).
- **Key Commands:**
    - CREATE: To create new tables, schemas, indexes, or databases.
    - ALTER: To modify existing database structures.

- DROP: To delete tables or databases.
- TRUNCATE: To remove all rows from a table without logging individual row deletions.

Data Languages:

1. **DML (Data Manipulation Language)**

- **Purpose:** Used to interact with and manipulate data stored in the database.
- **Key Commands:**
  - INSERT: To add new rows to a table.
  - UPDATE: To modify existing data.
  - DELETE: To remove specific rows from a table.
  - SELECT: To query and retrieve data.

2. **DCL (Data Control Language)**

- **Purpose:** Used to control access to data and manage database security.
- **Key Commands:**
  - GRANT: To provide access or privileges to users.
  - REVOKE: To remove access or privileges from users.

3. **TCL (Transaction Control Language)**

- **Purpose:** Used to manage transactions within the database, ensuring data integrity and consistency.
- **Key Commands:**
  - COMMIT: Saves changes made in a transaction to the database.
  - ROLLBACK: Reverts changes made in a transaction.
  - SAVEPOINT: Sets a point within a transaction to which you can roll back later.

4. **DRL/DSL (Data Retrieval Language or Data Query Language)**

- **Purpose:** Used to fetch data from the database.
- **Key Command:** SELECT (though part of DML, it is often categorized separately as DRL or DSL).

Phantom Reads:

occur in the context of database transactions and are one of the anomalies that arise due to concurrent access to a database. They happen when a transaction retrieves a set of rows based on a specific condition, and another transaction inserts or deletes rows that match the same condition before the first transaction completes. As a result, subsequent queries in the first transaction yield different results, even though it hasn't made any changes itself.

Functions

A **function** is a reusable, named block of SQL code that takes input parameters, performs a calculation or operation, and returns a single value or result.

Key Features:

- Returns a value (e.g., scalar, table, or other data types).
- Can take input parameters.
- Used for computations, transformations, or reusable logic in queries.
- Called within SQL statements like SELECT.

Stored Procedures

A **stored procedure** is a precompiled block of SQL code that can perform multiple operations, including modifying data. It is generally used to execute a sequence of SQL statements.

Key Features:

- Does **not** necessarily return a value (though it can).
- Can take input, output, or input-output parameters.
- Used for complex business logic, data processing, or batch operations.
- Invoked using the CALL or EXEC command.

Triggers

A **trigger** is a special type of stored procedure that automatically executes in response to specific database events (e.g., INSERT, UPDATE, DELETE) on a table.

Key Features:

- Does not take parameters.
- Executes automatically based on a defined event.
- Can be used for data validation, auditing, or enforcing business rules.

ACID properties:

Atomicity

- **Definition**: A transaction is treated as a single, indivisible unit. Either all operations within the transaction are completed successfully, or none of them are applied.
- **Purpose**: Prevents partial updates to the database.
- **Example**: In a bank transfer, if money is debited from one account but not credited to another, the transaction will roll back to maintain consistency.

Consistency

- **Definition**: A transaction ensures that the database remains in a valid state before and after the transaction.
- **Purpose**: Enforces defined rules, such as constraints, so the database doesn't violate its integrity.
- **Example**: If a transaction violates a foreign key constraint, it will not be committed.

Isolation

- **Definition**: Transactions are executed independently of one another, preventing interference from concurrent transactions.
- **Purpose**: Avoids issues like dirty reads, phantom reads, and non-repeatable reads.
- **Example**: If two users update the same record simultaneously, isolation ensures one update doesn't overwrite the other improperly.

Durability

- **Definition**: Once a transaction is committed, its changes are permanently stored in the database, even in case of a system crash.
- **Purpose**: Ensures data is not lost after a successful transaction.
- **Example**: After a bank transfer is completed and the system crashes, the transaction will still be saved when the system recovers.

What Are Indexes in Databases?

An **index** in a database is a data structure that improves the speed of data retrieval operations on a table. It acts like a "shortcut" or "table of contents" for database records, allowing queries to locate data more quickly without scanning the entire table.

Importance of Indexes

1. **Faster Query Performance**
    - Indexes significantly reduce the time taken to find rows matching a query by allowing the database to search more efficiently.
2. **Facilitates Joins**
    - When joining tables, indexed columns allow the database to find and match related rows quickly.
3. **Supports Uniqueness**
    - Unique indexes enforce the uniqueness of values in a column (e.g., primary key or unique constraints).
4. **Improves Scalability**
    - For large datasets, indexes ensure that applications perform consistently, even as the data grows.
5. **Full-Text Search**
    - Special types of indexes allow for efficient searching within text fields, enabling capabilities like keyword-based searches.

Sharding:

In **DBMS (Database Management Systems)**, **sharding** refers to the partitioning of a database into smaller, manageable pieces (shards) to improve performance and scalability. Sharding can be done either **vertically** or **horizontally**, based on how the data is divided.

Vertical Sharding

- **Definition**: Vertical sharding divides the database by columns (attributes), storing different sets of columns in separate shards.
- **How It Works**: Each shard contains fewer attributes, but all rows for those attributes.

Horizontal Sharding

- **Definition**: Horizontal sharding divides the database by rows (records), storing different sets of rows in separate shards.
- **How It Works**: Each shard contains all columns but only a subset of rows.