18/01/2025

Database Fundamentals

Database Keys

- **Primary Key**: A unique identifier for a record in a table. It ensures that no two rows have the same value in the primary key column(s).
- **Foreign Key**: A field (or a combination of fields) in one table that refers to the primary key in another table, creating a relationship between the two tables.
- **Unique Key**: Ensures that all values in a column are unique, but unlike the primary key, it can allow null values.

SQL Operations

- **CRUD**: Basic operations that can be performed on data in a database.
    - **Create**: Inserting new records into the database.
    - **Read**: Querying data from the database.
    - **Update**: Modifying existing records in the database.
    - **Delete**: Removing records from the database.

Benefits of RDBMS (Relational Database Management Systems)

1. **Structured Data Organization**: RDBMS uses tables with rows and columns to organize data, making it easy to manage and query.
2. **Data Integrity (ACID properties)**:
    - **Atomicity**: All or nothing principle in transactions.
    - **Consistency**: Transactions bring the database from one valid state to another.
    - **Isolation**: Ensures that concurrently executing transactions do not interfere with each other.
    - **Durability**: Once a transaction is committed, it is permanent, even in the case of a system crash.
3. **Data Relationships**: RDBMS allows data to be related across multiple tables through keys, enabling complex queries and ensuring data consistency.
4. **SQL Support**: SQL is a standardized language for managing and querying relational databases.
    - **DDL (Data Definition Language)**: Defines and manages the structure of database objects (e.g., CREATE, ALTER, DROP).
    - **DML (Data Manipulation Language)**: Manages data within schema objects (e.g., SELECT, INSERT, UPDATE, DELETE).
    - **DCL (Data Control Language)**: Deals with permissions and access control (e.g., GRANT, REVOKE).
    - **DQL (Data Query Language)**: Deals with data queries (e.g., SELECT).
    - **TCL (Transaction Control Language)**: Manages transaction control (e.g., COMMIT, ROLLBACK).
5. **CQRS (Command Query Responsibility Segregation)**: CQRS separates read and write operations into different models, which can improve scalability and performance in complex systems.
6. **Scalability**: RDBMS has limitations in horizontal scaling (e.g., cannot scale as easily as NoSQL databases), though it can scale vertically.

Phantom Reads

- **Phantom Reads**: A phenomenon in databases where a transaction reads a set of rows that match a certain condition, but another transaction inserts or deletes rows that match the condition during the execution of the first transaction.
- **Prevention**: Higher isolation levels like **Serializable** prevent phantom reads.

Serializable Read vs Repeatable Read

Repeatable Read:

- **Prevents**: Dirty reads (reading uncommitted data) and non-repeatable reads (where data changes between reads within the same transaction).
- **Allows**: Phantom reads (new rows can be inserted that match the condition in the query).
- **How it works**: It ensures that once a row is read, its value cannot change for the duration of the transaction. However, new rows can still be inserted that satisfy the condition of the query.

Serializable Read:

- **Prevents**: Dirty reads, non-repeatable reads, and phantom reads.
- **How it works**: This level ensures complete isolation by serializing transactions, meaning no other transactions can modify or insert data that could affect the transaction's result. It uses locking mechanisms to enforce this.

Key Differences:

- **Locking**: Serializable Read uses more extensive locking than Repeatable Read.
- **Performance**: Serializable Read has a greater performance overhead due to stricter locking and isolation.
- **Phantom Reads**: Serializable Read prevents phantom reads, while Repeatable Read allows them.

CAP Theorem

- **Consistency**: Every read returns the most recent write (or an error).
- **Availability**: Every request receives a response, either success or failure.
- **Partition Tolerance**: The system continues to function despite network partitions or communication breakdowns.

The CAP theorem suggests that a distributed system can only guarantee two of the three properties at a time:

1. **CA (Consistency + Availability)**: No network partition, but you may experience slower responses.
2. **CP (Consistency + Partition Tolerance)**: Consistent reads/writes despite partitions, but may reduce availability.
3. **AP (Availability + Partition Tolerance)**: The system remains available despite partitions, but the consistency of data may be compromised.

21/01/2025

Isolation Level Types

1. **Read Uncommitted (Dirty Read):**
    - Definition: This isolation level allows transactions to read data that has been modified but not yet committed by other transactions.
    - Advantages: It provides the highest level of concurrency and can be faster.
    - Disadvantages: The major risk is that it can read "dirty" data, meaning data that might later be rolled back, leading to inconsistencies or incorrect results.
2. **Read Committed:**
    - Definition: A transaction can only read data that has been committed by other transactions. It cannot read uncommitted changes.
    - Advantages: Ensures that the transaction only sees valid, committed data.
    - Disadvantages: It still allows for non-repeatable reads (i.e., a value can change between two reads in the same transaction), which may cause inconsistencies in some use cases.

Advantages of RDBMS

1. **Structured Data:** Data is organized into tables, which allows for easy querying and reporting.
2. **Data Integrity:** RDBMS enforces data constraints like primary keys, foreign keys, and unique constraints, ensuring the integrity and validity of the data.
3. **Flexibility:** Supports powerful SQL queries and complex joins across tables, allowing for sophisticated data manipulation.
4. **ACID Compliance:** RDBMS ensures that transactions are Atomic, Consistent, Isolated, and Durable, providing reliable data handling even in case of failures.
5. Standardized: SQL is a standardized language, making it easier to work with RDBMS across different platforms.
6. Security: Offers a well-established security model to control user access and permissions.

Disadvantages of RDBMS

1. Scalability: Horizontal scaling (scaling out across multiple servers) can be difficult and expensive. RDBMS tends to struggle with very large datasets or high-throughput systems.
2. Complexity in Large Data Models: As data grows, the schema can become complex, leading to performance degradation in large-scale operations.
3. Limited Flexibility: Changes to the database schema (e.g., adding or altering tables) can require downtime or complex migrations.
4. Performance: Complex queries with large joins can lead to performance issues, particularly when working with huge volumes of data.

5. Cost: Commercial RDBMS can be expensive, especially when licensing costs are involved.

Sharding: Distribution of Data

Sharding refers to the practice of distributing data across multiple databases or servers, called "shards," to improve scalability and performance.

Goal: To break large databases into smaller, more manageable pieces to handle increased load and reduce bottlenecks.

Vertical Sharding

Vertical sharding involves splitting a database by tables or columns, rather than rows.

Example: A user database might be vertically split where one shard stores user data (name, email), and another shard stores user activity logs (logins, posts).

Advantages: It can help with performance optimization as each shard contains fewer and more specialized tables.

Disadvantages: Can lead to complex query management, and may cause data retrieval challenges if queries need to access multiple shards.

Normalization

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It divides large tables into smaller, related tables and establishes relationships between them.

1. **1NF (First Normal Form):**

   Rule: A table is in 1NF if it has no repeating groups or arrays. All columns must contain atomic (indivisible) values.

   Purpose: Ensures that each record and field is unique, and no column contains multiple values.

   Example: A table storing student information should have separate rows for each student, rather than grouping multiple subjects into a single column.

2. **2NF (Second Normal Form):**

   Rule: A table is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key.

   Purpose: Eliminates partial dependency, where non-key attributes depend only on part of the composite primary key.

   Example: If a table has a composite key (StudentID, CourseID) but a non-key attribute depends only on StudentID (not CourseID), this dependency violates 2NF and should be removed by creating a new table.

3. 3NF (Third Normal Form):

Rule: A table is in 3NF if it is in 2NF and all non-key attributes are non-transitively dependent on the primary key.

Purpose: Removes transitive dependencies (i.e., when a non-key attribute depends on another non-key attribute).

Example: If a table stores employee information and also stores the department manager's name, this introduces a transitive dependency that should be split into two tables (one for employees and one for departments).

4. 4NF (Fourth Normal Form):

Rule: A table is in 4NF if it is in 3NF and has no multi-valued dependencies.

Purpose: Eliminates situations where multiple independent multi-valued facts are stored in the same table.

Example: A table where an employee has multiple skills and certifications should split into two separate tables to avoid storing both in a single record.