What is normalization? Explain its types.

Answer: Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. The main types are:

- **First Normal Form (1NF):** Ensures that the table has only atomic (indivisible) values and each column contains unique values.
- **Second Normal Form (2NF):** Achieves 1NF and ensures that all non-key attributes are fully functionally dependent on the primary key.
- **Third Normal Form (3NF):** Achieves 2NF and ensures that all attributes are not only fully functionally dependent on the primary key but also independent of each other (no transitive dependency).

How do you optimize a query in MySQL?

- **Use Indexes:** Ensure that appropriate columns are indexed.
- **Avoid SELECT *:** Select only the columns you need.
- **Use LIMIT:** Restrict the number of rows returned.
- **Optimize Joins:** Ensure joins are performed on indexed columns and use appropriate join types.
- **Use EXPLAIN:** Analyze query performance using EXPLAIN to understand how MySQL executes the query.
- **Optimize WHERE Clauses:** Use efficient conditions in WHERE clauses to limit the number of rows scanned.
- **Avoid Subqueries:** Use joins instead of subqueries where possible.

Explain the ACID properties in the context of MySQL.

Answer: ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable processing of database transactions:

- **Atomicity:** Ensures that all operations within a transaction are completed successfully. If any operation fails, the entire transaction fails.
- **Consistency:** Ensures that a transaction brings the database from one valid state to another, maintaining database invariants.
- **Isolation:** Ensures that concurrently executing transactions do not affect each other.

- **Durability:** Ensures that once a transaction is committed, the changes are permanent, even in the event of a system failure.

What are stored procedures and functions in MySQL?

Stored Procedure: A set of SQL statements that can be stored and executed on the database server. They can accept input parameters and perform complex operations.

```sql
CREATE PROCEDURE procedure_name (IN param1 INT, OUT param2 INT)
BEGIN
    -- SQL statements
END;
```

Function: Similar to stored procedures but returns a single value and can be used in SQL expressions.

```sql
CREATE FUNCTION function_name (param1 INT) RETURNS INT
BEGIN
    -- SQL statements
    RETURN some_value;
END;
```

Explain the concept of replication in MySQL.

Answer: Replication is the process of copying data from one MySQL server (the master) to one or more MySQL servers (the slaves). It provides redundancy and high availability. There are different types of replication, such as:

- **Asynchronous Replication:** The master does not wait for the slave to acknowledge the receipt of data.
- **Semi-Synchronous Replication:** The master waits for at least one slave to acknowledge the receipt of data.
- **Synchronous Replication:** All changes must be confirmed by all slaves before the master applies the changes (rarely used due to performance issues).

What are the common storage engines used in MySQL?

- **InnoDB:** Default storage engine, supports ACID-compliant transactions, foreign keys, and row-level locking.
- **MyISAM:** Older engine, does not support transactions or foreign keys, but is faster for read-heavy operations.
- **MEMORY:** Stores data in memory for fast access, but data is lost when the server is restarted.
- **CSV**: Stores data in CSV files, useful for data exchange with other applications.

UNION and UNION ALL

The main difference between UNION and UNION ALL is that UNION removes duplicate rows, while UNION ALL does not:

Database sharding and partitioning

Database sharding and partitioning are techniques used to manage large datasets by breaking them into smaller, more manageable pieces, which can help improve performance, scalability, and maintainability. Although they share similar goals, they are different in their implementation and use cases.

Database Sharding

Sharding involves splitting a large database into smaller, distinct databases, each holding a portion of the data. This technique is especially useful for horizontally scaling out databases to distribute the load across multiple servers.

Key Characteristics:

- **Horizontal Partitioning:** Data is divided across multiple databases (shards) based on a shard key, such as a user ID or geographical region.
- **Independent Databases:** Each shard is a fully functional database with its own schema and data, typically located on different servers.
- **Increased Availability:** Sharding can improve availability by isolating failures to individual shards.
- **Scalability:** It allows for distributing the load and storage requirements across multiple servers.

Example:

Imagine a social media application with millions of users. Instead of storing all user data in a single database, you can shard the database based on user ID ranges:

- Shard 1: Users with IDs 1-1,000,000
- Shard 2: Users with IDs 1,000,001-2,000,000
- Shard 3: Users with IDs 2,000,001-3,000,000

This distribution ensures that each shard handles a smaller subset of the total user base, reducing load and improving performance.

Database Partitioning

Partitioning involves dividing a single large database table into smaller, more manageable pieces called partitions. Each partition is a subset of the table's data and is managed within the same database instance.

Key Characteristics:

- **Vertical Partitioning:** Data is divided into smaller, more manageable pieces within a single table, based on certain criteria such as range, list, hash, or composite methods.
- **Single Database Instance:** All partitions are managed by the same database instance.
- **Improved Query Performance:** Partitioning can improve query performance by allowing the database to scan only the relevant partitions rather than the entire table.

- **Easier Maintenance:** It simplifies maintenance tasks such as backup, archiving, and deletion of old data.

Types of Partitioning:

1. **Range Partitioning:** Divides data based on a range of values.

```
sql
Copy code
CREATE TABLE sales (
    sale_id INT,
    sale_date DATE,
    amount DECIMAL(10,2)
)
PARTITION BY RANGE (YEAR(sale_date)) (
    PARTITION p0 VALUES LESS THAN (2010),
    PARTITION p1 VALUES LESS THAN (2015),
    PARTITION p2 VALUES LESS THAN (2020),
    PARTITION p3 VALUES LESS THAN (2025)
);
```

2. **List Partitioning:** Divides data based on a predefined list of values.

```
sql
Copy code
CREATE TABLE users (
    user_id INT,
    country VARCHAR(50)
)
PARTITION BY LIST (country) (
    PARTITION usa VALUES IN ('USA'),
    PARTITION uk VALUES IN ('UK'),
    PARTITION india VALUES IN ('India')
);
```

3. **Hash Partitioning:** Divides data based on a hash function applied to a partition key.

```
sql
Copy code
CREATE TABLE orders (
    order_id INT,
    customer_id INT,
    order_date DATE
)
PARTITION BY HASH(customer_id) PARTITIONS 4;
```

4. **Composite Partitioning:** Combines multiple partitioning methods.

```
sql
Copy code
CREATE TABLE logs (
    log_id INT,
    log_date DATE,
    severity VARCHAR(10)
)
PARTITION BY RANGE (YEAR(log_date)) SUBPARTITION BY HASH (severity)
SUBPARTITIONS 4 (
```

```
        PARTITION p0 VALUES LESS THAN (2010),
        PARTITION p1 VALUES LESS THAN (2015),
        PARTITION p2 VALUES LESS THAN (2020),
        PARTITION p3 VALUES LESS THAN (2025)
    );
```

Comparison

- **Sharding:** Splits data across multiple databases (horizontal scaling), improving distribution and management across servers.
- **Partitioning:** Divides data within a single database table into smaller parts (vertical scaling), improving query performance and maintainability within the same database instance.

Use Cases

- **Sharding:** Ideal for applications requiring horizontal scaling, such as large-scale web applications with massive datasets that need to be distributed across multiple servers.
- **Partitioning:** Suitable for applications needing efficient query performance and easier maintenance for large tables within a single database, such as logging systems, data warehouses, and archival systems.

Understanding these techniques and their differences is crucial for designing and managing large-scale databases effectively.