**Problem Statement:** Explain how indexing affects **query performance** in a large system.

In large systems, indexing significantly improves query performance by allowing the database to quickly locate and access specific data, reducing the need for full table scans and leading to faster data retrieval and overall system responsiveness.

Indexing affects query performance in following ways

1. Faster data retrieval

2. Reduced full table scans

3. Optimized Searching

4. Increased storage space

5. Slower write operations

**Problem Statement:** Use EXPLAIN to analyze a complex query and suggest optimizations.

select passengers.name, passengers.passenger\_id, passengers.email,passengers.loyalty\_points,  
 bookings.booking\_id, bookings.passenger\_id, bookings.train\_id, bookings.journey\_date,  
 bookings.seat\_number, bookings.class, bookings.total\_fare,bookings.booking\_status  
 from passengers inner join bookings on passengers.passenger\_id=bookings.passenger\_id;

**Problem Statement:** Create a new MySQL user customer\_service with read-only access to bookings.

GRANT SELECT ON bookings TO 'customer\_service'@'localhost;'



**Problem Statement:** Implement SQL injection prevention in a query that retrieves train details.

**Problem Statement:** Discuss the role of GRANT, REVOKE, and ROLE-based access control for staff users.

**GRANT:** To control what actions a user can perform, the GRANT statement is essential for assigning specific privileges to user accounts.

**REVOKE:** Revoke is a command used to remove privileges to user accounts.

**Problem Statement:** Show how to perform a full backup of the train reservation database using mysqldump.

mysqldump –u root –p train\_system.sql > C:\Desktop\train\_system.sql

**Problem Statement:** Explain the need for **incremental backups** for high-traffic systems.

The sole purpose of incremental file backup is to capture any changes that have happened since the last archive. An incremental backup process compares current data against a prior full backup. If an incremental backup is taken before a full backup is performed, the backup system forces a full backup instead. A full backup covers all folders and files throughout a system.

**Problem Statement:** Describe **point-in-time recovery** using MySQL binary logs.

Suppose you deleted an important database table at 2 p.m. on a Wednesday. You realize this fifteen minutes later and you need to restore the data. The table in the replica has also been deleted.However, if you back up your data at 1 a.m. every day, the closest recovery point to when the table was deleted will be at 1 a.m. that Wednesday. When you restore data, you will lose 13 hours of data. But if you use the *point-in-time recovery* strategy, you can recover data as of 1:55 pm, **losing only 5 minutes!.** MySQL has a feature known as *binary logs*. Binary logs are files containing events that change the database. These files have a binary format,but can be converted to SQL instructions and re-run. Therefore, you can do a *baseline full backup* with *mysqldump*, and set up a process to copy all binary log files created after the baseline backup to the destination

**Problem Statement:** Explain how **MySQL replication** improves performance.

MySQL replication improves performance by distributing read queries across multiple slave servers, thereby reducing the load on the primary (master) server and enabling horizontal scaling for increased capacity.

Discuss **partitioning** strategies for large tables with millions of records.

For large tables with millions of records, partitioning strategies involve dividing the table into smaller, more manageable chunks based on a chosen key, enhancing query performance and data management. Common strategies include range, list, and hash partitioning.

1.] Partitioning Strategies:

Range Partitioning: Divides data based on a range of values in a specific column, often used for date-based data or other ordered data.

Example: Partitioning a table of sales transactions by year or month.

List Partitioning: Divides data based on a list of values in a specific column, useful when you have a predefined set of categories or types.

Example: Partitioning a product table by product category.

2] Hash Partitioning: Divides data randomly based on the hash value of a column, useful for distributing data evenly across partitions.

Example: Partitioning a user table by user ID.

**Problem Statement:** Describe how sharding can distribute train data across multiple servers.

Sharding distributes train data across multiple servers by partitioning the dataset into smaller, independent subsets and assigning each shard to a specific server or node, allowing for horizontal scalability and improved performance.

How Data Partitioning is done: Sharding involves dividing a large dataset into smaller, manageable chunks, called shards. Each shard is then assigned to a different server or node in the distributed system. The data is partitioned based on a specific criterion, such as a range of keys, hash values, or other data attributes.

For example, you could shard a user database based on user IDs, with each shard containing a range of user IDs.