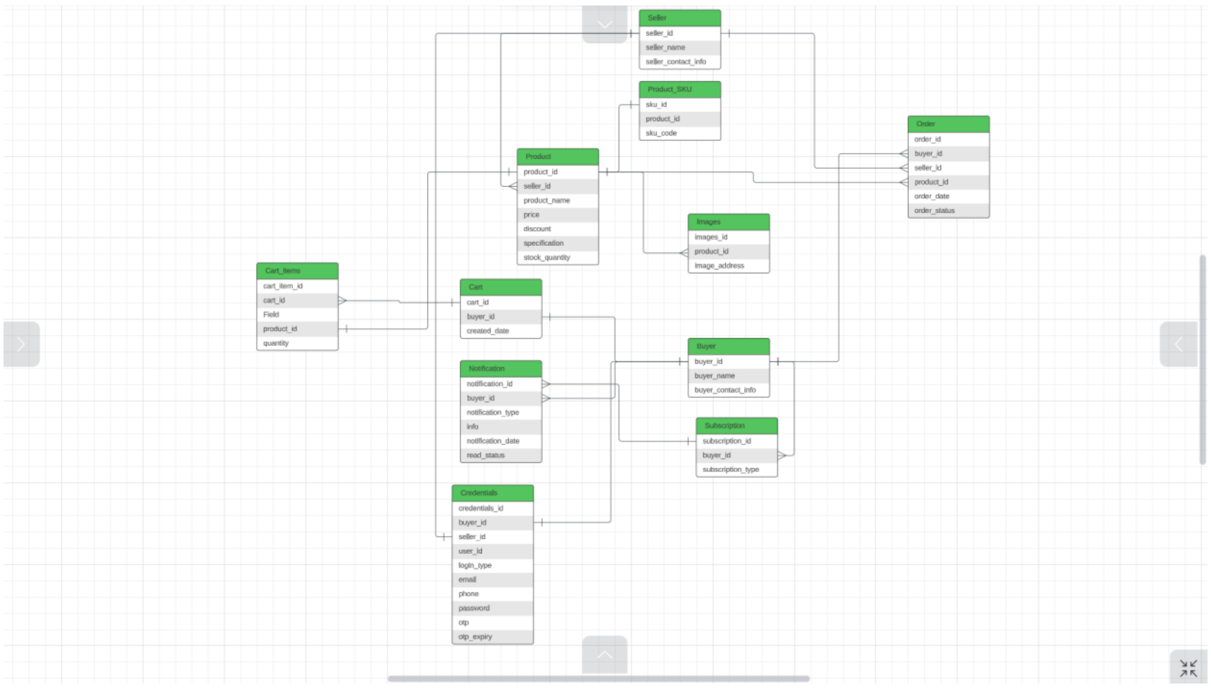
Deliverables

1. Physical Entity Relationship diagram of database.



2. Explain about searching performance. How will you handle replication in SQL for searching &

Reporting?

Given that we have 100K users concurrently accessing the database, our focus should be on ensuring high performance, scalability, and availability. To achieve these goals, a combination of replication techniques would be most suitable. Here's some recommended approaches:

1. Transactional Replication

2. Read-Only Replicas

3. Multi-Region Replication

4. High Availability and Disaster Recovery

5. Cache Mechanisms

It's essential to continuously monitor the performance and health of the replication setup, particularly as the user base grows. Regularly conduct load testing and performance tuning to ensure that the replication strategy meets the demands of your expanding user concurrency.

Sharding or partitioning the database to distribute the load across multiple physical servers or instances can help handle the increasing number of concurrent users more effectively.

3. Explain what major factors are taken into consideration for performance.

Improving the overall performance of a database design involves optimizing various aspects to ensure efficient data retrieval, manipulation, and storage. Here are several factors that can improve database performance:

1. Indexing

2. Query Optimization

3. Normalization

4. Denormalization

5. Caching

6. Partitioning

7. Hardware and Infrastructure

8. Connection Pooling

9. Load Balancing

10. Replication

11. Monitoring and Tuning

12. Archiving and Purging

13. Asynchronous Processing

4. Mention about Indexing, Normalization and Denormalization.

Indexing: In our design we can use indexing of the columns that are frequently queried and large in size. For Example, Buyers/Sellers can check their order details based on order date so there is filtering required based on order_date field. So, we can apply indexing on the same attribute.

Normalization: Normalization is generally done to reduce redundancy of data while making sure that it does not effect query performance. Like in our design we have multiple images for a product. It could have been messy to put all images in the product table as there can be different number of images for different products. So here we have to create a new table to avoid any redundancy.

Denormalization: Suppose, we want to display a list of products with their names, categories, and total sales quantity. Here we have a complex query that might involve complex joins and hence increase the query time. To address this, we can denormalize the Products table by adding an additional column for total_quantity_sold.

5. How will you handle scaling, if required at any point of time.

Following are several strategies you can employ to ensure our database can scale effectively:

Vertical Scaling (Scaling Up): Initially, we can scale our database vertically by upgrading the hardware of our database server. This involves increasing the server's CPU, RAM, and storage capacity to handle additional load. While this approach provides immediate capacity improvements, there is a limit to how much you can scale vertically.

Horizontal Scaling (Scaling Out): When vertical scaling reaches its limits or becomes cost-prohibitive, we can consider horizontal scaling. In this approach, we distribute the database workload across multiple servers or instances. Some common techniques for horizontal scaling include:

Sharding

Replication

Load Balancing

Caching

Asynchronous Processing

Cloud Services

6. Mention all the assumptions you are taking for solutions.

Here are the assumptions made for the above solutions:

1. Simplified Scope: The provided solutions are based on a simplified scope of an E-commerce application, focusing on specific modules (Inventory, Order/Cart, Notification, and Authentication) and their relevant features.

2. Database Structure: The database schema and relationships are designed to demonstrate the connections between different modules. In a real-world scenario, additional entities and attributes might be needed to fully represent the application's requirements.

3. Data Types: Data types for attributes are suggested based on general conventions, but the specific data types used in the actual implementation may vary depending on the database system and application requirements.

4. Security Considerations: While some security aspects are mentioned, such as hashing passwords and user authentication, the provided solutions do not cover all security measures needed for a production-grade E-commerce application. Implementing comprehensive security measures is essential in a real-world scenario.

5. Caching and Optimization: Specific caching mechanisms and optimization techniques are not fully detailed in the solutions. In practice, various caching strategies and performance optimization methods should be carefully considered and implemented based on the application's unique needs.

6. Assumption of Normalization/Denormalization: The level of normalization and denormalization in the database design is assumed based on the provided scope. In practice, the level of normalization should be determined after a thorough analysis of the data and application requirements.

7. Scalability Details: The solutions mentioned scalability strategies like horizontal scaling and load balancing. However, the specific implementation details and tools may vary depending on the technology stack and infrastructure used.