DATABASE DESIGN ASSIGNMENT

Sagar Khatri

NAGARRO

Contents

Database Design Assignment

E-Commerce Application

Scope is limited to following Modules and corresponding features.

Inventory Module

1. The seller can add products, images, and details of product ex: Price, discount, specification, Product SKUs etc.
2. At a time, there can be 100K users (active in application) searching for a product.

Order/Cart Module

1. Buyers can add products to cart and buy them.
2. Seller can accept order or reject order.
3. Buyers/Sellers can check their order details based on order date.

Notification Module

1. Buyer will get notification for product availability if he/she subscribed to same.
2. The buyer will get notification about product orders and order status when ready to ship, packed and delivered etc.
3. Application should support push notifications for promotional offers and sale discounts.

Authentication & Authorization

1. Buyer and Seller should be able to login via email/password or Phone number with OTP.
2. Application should support Forget Password feature.

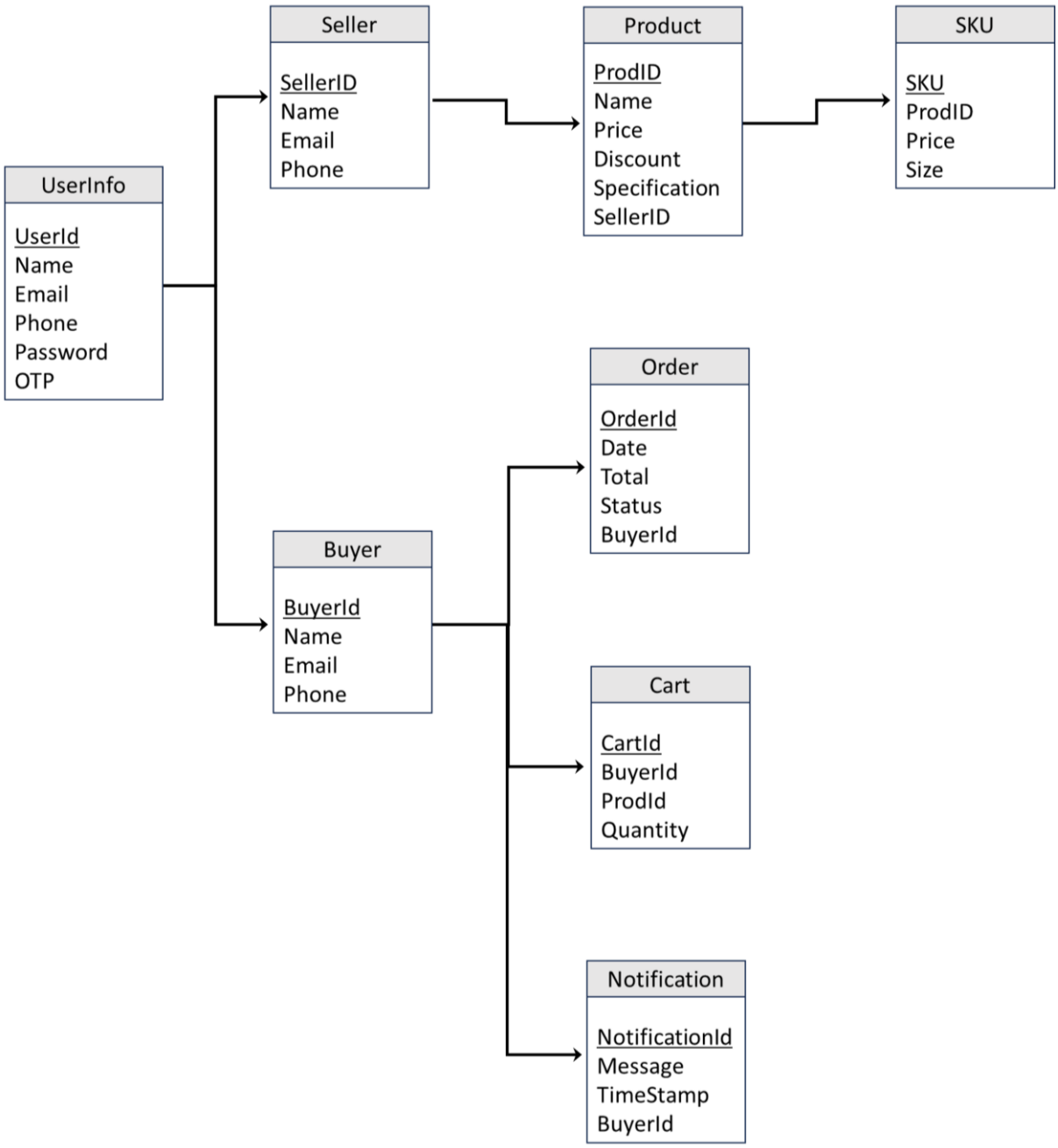
Consider above mentioned features, design physical entity relationship diagram. Use normalization wherever you think needed.

Database choice can be different for different modules.

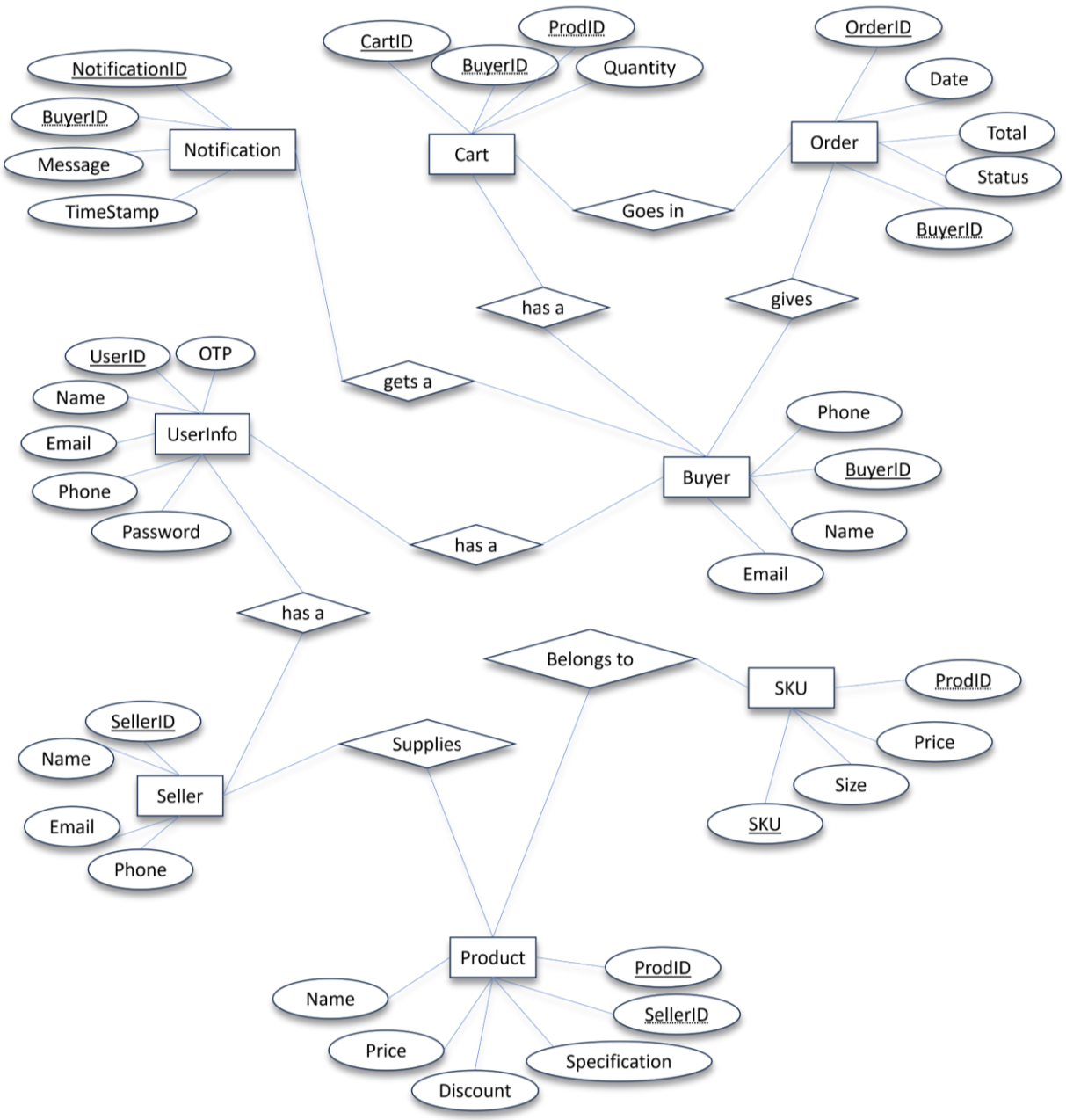
Deliverables

1. Physical Entity Relationship diagram of database.
2. Explain about searching performance. How will you handle replication in SQL for searching & Reporting?
3. Explain what major factors are taken into consideration for performance.
4. Mention about Indexing, Normalization and Denormalization.
5. How will you handle scaling, if required at any point of time.
6. Mention all the assumptions you are taking for solutions.

Database Diagram for E-commerce App



Entity-Relationship Diagram for E-commerce App



Relationships in the ER diagram for E-commerce App

Inventory Module:

- Seller (1) ---< Product (1 to Many)
- Product (1) ---< SKU (1 to Many)

Order/Cart Module:

- Buyer (1) ---< Order (1 to Many)
- Buyer (1) ---< Cart (1 to Many)
- Seller (1) ---< Order (1 to Many)

Notification Module:

- Buyer (1) ---< Notification (1 to Many)

Authentication & Authorization:

- UserInfo (1) ---< Buyer/Seller (1 to Many)

Explanation of ER diagram for E-Commerce App

Inventory Module:

- **Seller entity** represents the sellers in the system who can add products to sell.
- **Product entity** stores the details of products added by sellers, and each product can have multiple SKUs (Stock Keeping Units).
- **SKU entity** represents the specific variations of a product, each with its own SKU identifier, price, and size.

Order/Cart Module:

- **Buyer entity** represents the buyers in the system who can add products to their cart and place orders.
- **Order entity** stores the details of orders placed by buyers, including the order date, total amount, and order status (e.g., accepted, rejected).
- **Cart entity** holds the information about the products added to the cart by buyers, including the quantity.

Notification Module:

- **Notification entity** is used to send notifications to buyers about product availability, order updates, and promotions.
- Each notification is associated with a specific buyer and contains a message and timestamp.

Authentication & Authorization:

- **UserInfo entity** represents both buyers and sellers who can log in to the application using email/password or phone number with OTP (One-Time Password).
- **UserInfo entity** stores the login credentials (password and OTP) and other user information.

Question: Explain about searching performance. How will you handle replication in SQL for searching & Reporting?

Searching performance is a critical aspect of any E-Commerce application, as it directly impacts the user experience and overall system efficiency. Here are some considerations and strategies to improve searching performance:

1. **Indexing:** Creating appropriate indexes on the columns frequently used in search queries can significantly speed up the search process. Indexes act as data structures that allow the database to quickly locate rows based on the indexed columns. For example, indexing on product name, SKU, or category can speed up product searches.
2. **Query Optimization:** Writing efficient queries is crucial for good search performance. Ensure that queries use the right indexes and avoid unnecessary joins and subqueries. Use database tools to analyze query execution plans and identify areas for optimization.
3. **Caching:** Implement caching mechanisms to store frequently accessed data in memory, reducing the need to perform repeated database searches. Caching can be done at various levels, such as application-level caching or using caching solutions like Redis.
4. **Full-Text Search:** For complex search requirements, consider using full-text search capabilities provided by the database or dedicated search engines like Elasticsearch. Full-text search enables more advanced search capabilities, including keyword matching, relevance ranking, and partial matching.
5. **Sharding and Partitioning:** If the database grows large, consider sharding or partitioning data across multiple servers or databases. This technique horizontally distributes data, enabling more efficient search operations, especially in scenarios with massive amounts of data.

Overall, implementing proper indexing, query optimization, caching, and database replication strategies can significantly enhance searching performance and support reporting operations in an E-Commerce application.

Question: Explain what major factors are taken into consideration for performance.

Several major factors are taken into consideration for performance optimization in a database system:

- **Indexing:** Indexes are data structures that speed up data retrieval by allowing the database to locate rows quickly based on indexed columns. Choosing the right columns to index is critical for improving query performance. Too many indexes can slow down data modifications (inserts, updates, deletes), so a balance must be struck between query optimization and data modification efficiency.

- **Query Optimization:** Optimizing SQL queries is essential for efficient database performance. It involves crafting queries in a way that utilizes indexes, reduces the number of joins, minimizes subqueries, and avoids unnecessary data retrieval. Query optimization helps reduce the execution time of queries and prevents resource contention.

- **Data Modelling:** A well-designed data model can enhance database performance. Proper normalization and denormalization strategies should be employed to prevent data redundancy and improve data retrieval efficiency. The data model should also consider the application's specific requirements and access patterns.

- **Caching:** Caching frequently accessed data in memory can significantly reduce the need for repetitive database queries. Caching mechanisms such as application-level caching or using dedicated caching systems like Redis can improve response times and relieve database load.

- **Replication and Load Balancing:** Database replication and load balancing distribute the database workload across multiple servers, enhancing both read and write performance. Read replicas and load balancers ensure that read-intensive operations are distributed among replicas, reducing the load on the primary database.

- **Connection Pooling:** Maintaining a pool of pre-established connections to the database can avoid the overhead of frequently opening and closing connections, resulting in better performance and resource utilization.

- **Hardware and Infrastructure:** The underlying hardware and infrastructure play a significant role in database performance. Factors such as CPU, RAM, disk speed, and network connectivity impact the database's ability to handle concurrent users and process queries efficiently.

- **Database Configuration:** Optimizing database configuration parameters, such as buffer sizes, query cache settings, and concurrency levels, is crucial for performance. Each database system has specific configuration options that need to be adjusted according to the application's requirements.

- **Monitoring and Profiling:** Regular monitoring and profiling of the database system help identify performance bottlenecks, query hotspots, and resource utilization. Monitoring tools provide insights into the system's health and aid in proactive performance tuning.

- **Scaling Strategies:** As the application grows, scaling strategies should be in place to handle increased traffic and data volume. Scaling can be achieved through vertical scaling (upgrading hardware) or horizontal scaling (adding more servers or using sharding).

- **Backup and Recovery:** Proper backup and recovery strategies ensure data integrity and provide failover options in case of hardware failures or data corruption.

Question: What is meant by Indexing, Normalization and Denormalization?

Indexing, normalization, and denormalization are database design concepts and techniques used to optimize data storage, retrieval, and maintenance in relational databases.

1. Indexing:

- Indexing is a data structure technique used to enhance the speed of data retrieval from a database table.
- It involves creating a separate data structure known as an index that contains a sorted copy of selected columns from the table.
- Indexes act as a quick reference to the actual data, allowing the database to find and access rows more efficiently based on the indexed columns.
- When a query involves searching, sorting, or joining on indexed columns, the database can use the index to locate the required data without performing a full-table scan.
- Indexes improve query performance, especially for large tables, by reducing the time needed to locate and retrieve data.
- However, creating indexes also consumes additional storage space and may lead to increased overhead during data modification (e.g., inserts, updates, deletes). Therefore, careful consideration is necessary when deciding which columns to index.

2. Normalization:

- Normalization is a database design process that organizes data into separate tables to eliminate data redundancy and improve data integrity.
- The goal of normalization is to ensure that each piece of data is stored only once (atomicity) and that the data is logically structured to minimize anomalies.
- The normalization process involves applying a set of rules known as normal forms (e.g., first normal form1NF, second normal form2NF, third normal form3NF, etc.).
- Each normal form has specific requirements, such as removing repeating groups, eliminating partial dependencies, and ensuring that each non-key attribute depends solely on the table's primary key.

- By normalizing data, databases achieve data consistency, prevent update anomalies, and make data maintenance more straightforward.

3. Denormalization:

- Denormalization is the opposite of normalization; it involves intentionally introducing redundancy into a database by combining normalized tables.
- The main objective of denormalization is to improve query performance, especially for read-heavy applications or complex queries involving multiple joins.
- Denormalization achieves performance gains by selectively reintroducing redundant data or precomputing data into the database.
- By denormalizing certain tables or adding calculated fields, the database can store aggregated information, reducing the need for complex joins and speeding up query execution.
- However, denormalization increases data storage requirements and may lead to data integrity issues if updates and inserts are not carefully managed.

In summary, indexing enhances data retrieval speed, normalization reduces data redundancy and maintains data integrity, and denormalization optimizes query performance at the cost of some data redundancy.

Question: *how will scaling, if required in a future point of time ?*

Scaling an E-Commerce application is essential to handle increased traffic, data volume, and concurrent user demands as the application grows. Here are some strategies for scaling the E-Commerce app:

Vertical Scaling:

Vertical scaling involves upgrading the hardware resources of the existing server to handle increased loads. This can include adding more powerful CPUs, increasing RAM, or using faster storage devices (e.g., SSDs). Vertical scaling is relatively straightforward and can provide an immediate boost in performance. However, it has limitations and might not be cost-effective for significant traffic increases.

Horizontal Scaling:

Horizontal scaling involves adding more servers to distribute the workload across multiple machines. This can be achieved by using load balancers that distribute incoming requests to different application servers. Horizontal scaling is highly effective in handling increased traffic and improving application performance. It is also more cost-effective than vertical scaling, as commodity hardware can be used. However, it requires careful consideration of data distribution and maintaining data consistency among multiple servers.

Auto-scaling:

Implement auto-scaling mechanisms that automatically add or remove resources based on traffic patterns and resource utilization. Cloud platforms like AWS, Azure, and Google Cloud offer auto-scaling capabilities that can automatically adjust the number of servers based on demand.

Database Sharding:

Database sharding is a technique used in horizontal scaling, where data is partitioned across multiple database servers (shards). Each shard handles a

subset of data, allowing the application to spread the database load evenly. Sharding requires careful planning and a well-defined strategy for partitioning data to maintain data integrity and minimize cross-shard queries.

Caching:

Implementing caching mechanisms can significantly reduce the load on the database and application servers. Use caching solutions like Redis or Memcached to store frequently accessed data in memory. Caching can speed up response times and reduce the number of database queries.

Content Delivery Network (CDN):

Utilize a CDN to distribute static content (e.g., images, CSS, JavaScript) to edge servers located near the users. CDN caching can reduce the load on the application servers and improve the delivery speed of static assets.

Asynchronous Processing:

Consider offloading non-time-critical tasks, such as generating reports or sending notifications, to background processes or message queues. This approach prevents the main application from becoming bottlenecked by long-running tasks.

Database Replication:

Use read replicas to distribute read-intensive operations across multiple database servers. Read replicas provide a scalable way to handle read-heavy workloads without impacting the primary database's performance.

Performance Monitoring:

Continuously monitor application and database performance to identify bottlenecks and areas for improvement. Use monitoring tools to gain insights into system health, resource usage, and response times.

Question *: Mention all the assumptions you are taking for the solution.*

For the solution, I'm assuming the following:

- Each product can have multiple SKUs, representing different variations of the same product (e.g., different sizes or colors).
- Buyers and sellers have a one-to-many relationship, meaning a seller can have multiple products, and a buyer can place multiple orders.
- Notifications are sent to buyers based on their subscriptions and order status updates.
- User authentication and authorization are essential for buyers and sellers to access their respective functionalities.