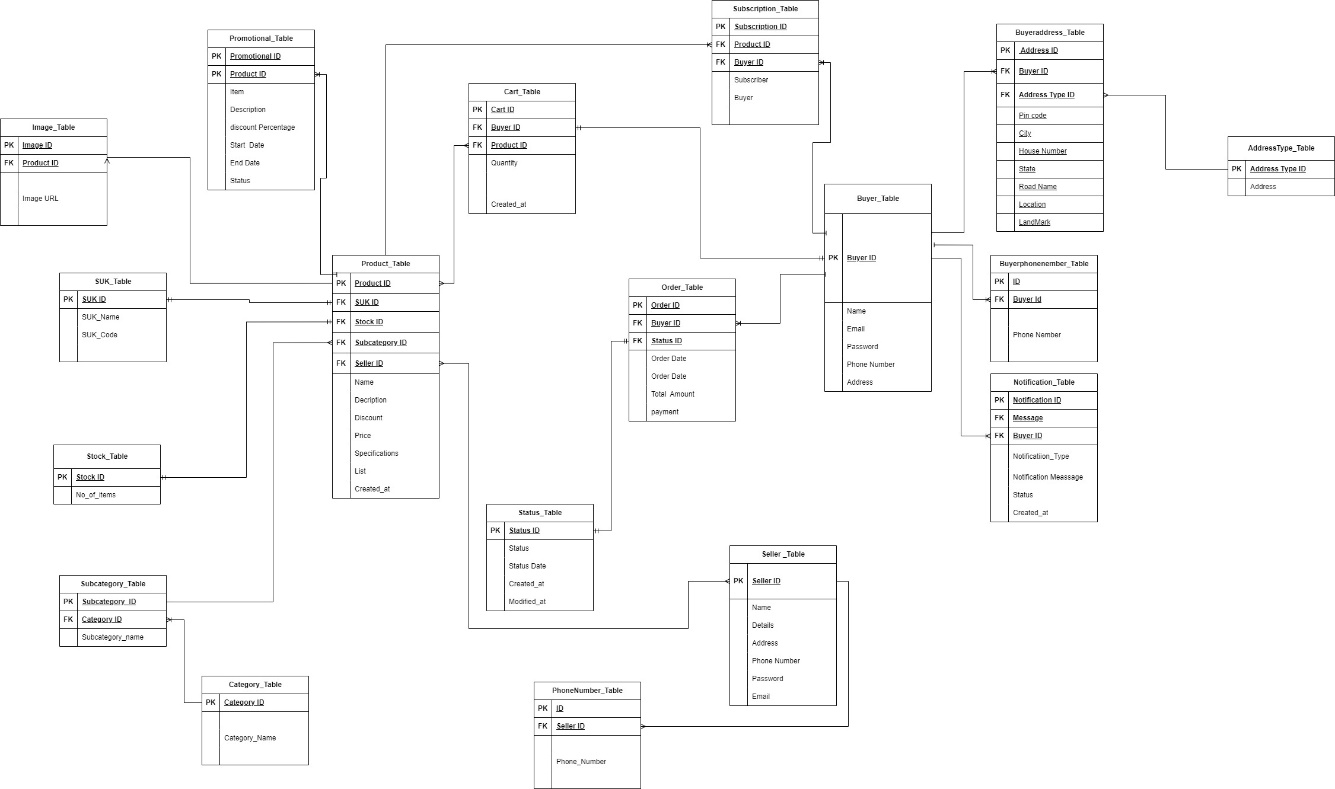
**Database Design Assignment**

1. **Physical Entity Relationship diagram of database**



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

**Searching Performance:**

The assignment focuses on improving searching performance through various strategies. Firstly, implementing caching mechanisms using tools like Memcached or Redis to store frequently accessed search results in memory, which reduces database load and enhances response times. Secondly, optimizing searching performance by creating appropriate full-text search indexes on commonly used columns like product name, description, and specification for fast and efficient search queries, especially during peak user activity. Thirdly, employing load balancing techniques to evenly distribute search and reporting queries across multiple database servers, preventing overloading and ensuring efficient resource utilization. Additionally, implementing replication in the SQL database through master-slave replication helps achieve high availability and load balancing, directing read queries to read replicas and distributing the load. Lastly, using materialized views to pre-compute and store complex query results, facilitating quick access for searching and reporting tasks with periodic updates to keep them synchronized with the underlying data. These strategies collectively enhance the overall performance and responsiveness of the application.

1. **Explain what major factors are taken into consideration for performance.**

1. Database Indexing: Proper indexing is crucial for efficient querying.  
Indexes speed up data retrieval based on specific columns or combinations of columns.  
Full-text search indexes can optimize searching performance for frequently used search fields.  
  
2. Hardware and Infrastructure:  
 Optimize server specifications, storage systems, network latency, and configuration for better performance.  
  
3. Regular Monitoring and Tuning:  
 Continuously monitor database performance and identify bottlenecks.  
 Apply performance tuning techniques like query optimization and index optimization.  
  
4. Database Connection Pooling:  
 Efficiently manage connections by reusing established connections.  
 Minimize overhead and improve response times and resource utilization.  
  
5. Scalability:  
 Ensure the database can handle increasing data and user traffic.  
 Use horizontal scaling (adding more servers) or vertical scaling (increasing server resources).  
  
6. Load Balancing:  
 Distribute workload across multiple servers to prevent performance bottlenecks.  
 Utilize hardware or software load balancers for even distribution.  
  
7. Query Optimization:  
 Analyse query execution plans and identify bottlenecks.  
 Implement query rewriting, caching, and tuning techniques.  
 Consider using stored procedures or prepared statements for frequently executed queries. Has context menuComposeParagraph

1. **Mention about Indexing, Normalization and Denormalization.**

**Normalization:-**

Normalization is a database design technique that eliminates redundant data and reduces data anomalies by breaking tables into smaller, more manageable entities. It helps ensure data integrity and reduces storage space.

1.Product, Product Category, and Subcategory tables:

These tables follow normalization principles by representing separate entities and avoiding data redundancy. The Product table contains product-specific information, the Product Category table stores the different categories of products, and the Subcategory table further categorizes the products within each category.

2.Promotion Offer table:  
 The Promotion Offer table represents a one-to-many relationship with the Product table. It follows normalization principles by having a separate table for storing promotion offers associated with each product. The product\_id serves as a foreign key to establish the relationship.

2.Order, Order Item, and Order Status tables:

These tables represent order-related information. The Order table stores general order details such as order date, total amount, and references the Buyer and Seller tables through foreign keys. The Order Item table represents the individual items within an order, with the order\_id and product\_id serving as foreign keys. The Order Status table holds different order statuses and their descriptions.

**Denormalization :-**

Denormalization, on the other hand, involves combining tables or adding redundant data to improve query performance and simplify data retrieval. Denormalization is often used in read-heavy applications to optimize data access. Depending on the specific requirements and performance considerations of the e-commerce application, certain denormalization techniques, such as using indexed views or materialized views, could be applied to improve query performance and reduce joins.

**Indexing :-**

Indexing is a technique used to optimize database performance by creating indexes that allow for faster data retrieval. Indexes are created on specific columns of database tables to speed up search queries. In the given scope, indexes can be created on columns that are frequently used in search operations, such as product names, description, and specification, buyer/seller email addresses, and order dates. Additionally, foreign key columns should be indexed for efficient joins between tables.

1.Primary Key columns:

Each table's primary key column should have an index by default. For example, the product\_id column in the Product table, the category\_id column in the Product Category table, etc. This indexing facilitates faster data retrieval and ensures uniqueness.

2.Foreign Key columns:

Columns used as foreign keys, such as the category\_id in the Product table, the buyer\_id in the Order table, etc., can be indexed to optimize joins and lookups. This indexing speeds up the process of retrieving data based on relationships between tables.

3.Frequently searched columns:

Columns that are frequently used in search conditions or filters, such as product\_name or order\_date, can be indexed. This indexing helps accelerate search queries and improves overall query performance.

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

Scalability is like having a flexible system that can adapt to changes in how many users are using application. It means the system can handle more or fewer users or data without breaking or becoming slow. So, whether your app becomes more popular and lots of people start using it, or if usage decreases, the system can adjust accordingly to keep things running smoothly.

To handle scaling in an e-commerce application, we can employ various strategies and technologies. Here are some approaches we can consider:

* Scalability ensures a flexible system that can handle varying numbers of users without performance issues.
* Two types of database scaling: vertical scaling (adding resources to a single machine) and horizontal scaling (adding more servers).
* Content Delivery Network (CDN) caches static content, reducing server load and improving response times globally.
* Auto-scaling adjusts the number of servers based on predefined metrics to meet demand dynamically.
* Caching mechanisms, like Redis or Memcached, reduce database load by storing frequently accessed data.
* Load balancing distributes network traffic across backend servers, enhancing redundancy and efficiency.
* Database replication ensures data consistency across servers; transactional replication is preferred for e-commerce applications handling many users.
* Snapshot replication, useful for read-heavy workloads with infrequent data changes.
* Merge replication, suitable for multiple subscribers updating a central database periodically.
* Peer-to-peer replication, syncing multiple databases in a peer-to-peer network for load balancing and high availability.

1. **Mention all the assumptions you are taking for solutions**

* Different modules in the e-commerce application can use different database management systems like MySQL, PostgreSQL, or MongoDB based on their specific requirements and strengths.
* The provided diagram is a simplified representation of entities and their relationships for the given scope, but additional entities and relationships may exist based on specific application requirements.
* For simplicity, a single database is considered in the diagram, and not all possible attributes for each entity are shown. Additional attributes and relationships can be added as needed.

The diagram assumes that the authentication and authorization functionality meets the given requirements, but specific security implementation details like encryption, secure communication protocols, and user roles/permissions are not depicted.

* + Token-based authentication or Identity server can be used instead of the depicted authentication method.
  + Product-Image: A one-to-many relationship exists between products and their multiple images.
  + Buyer-BuyerAddress: A one-to-many relationship exists between buyers and their multiple addresses.
  + Buyer-BuyerPhoneNumber: A one-to-many relationship exists between buyers and their multiple phone numbers.
  + BuyerAddress-AddressType: A many-to-many relationship exists between buyer addresses and their multiple address types.
  + Buyer-Cart: A one-to-one relationship exists between buyers and their carts.
  + Buyer-Order: A one-to-many relationship exists between buyers and their multiple orders.
  + Buyer-Subscription: A one-to-many relationship exists between buyers and their multiple subscriptions.