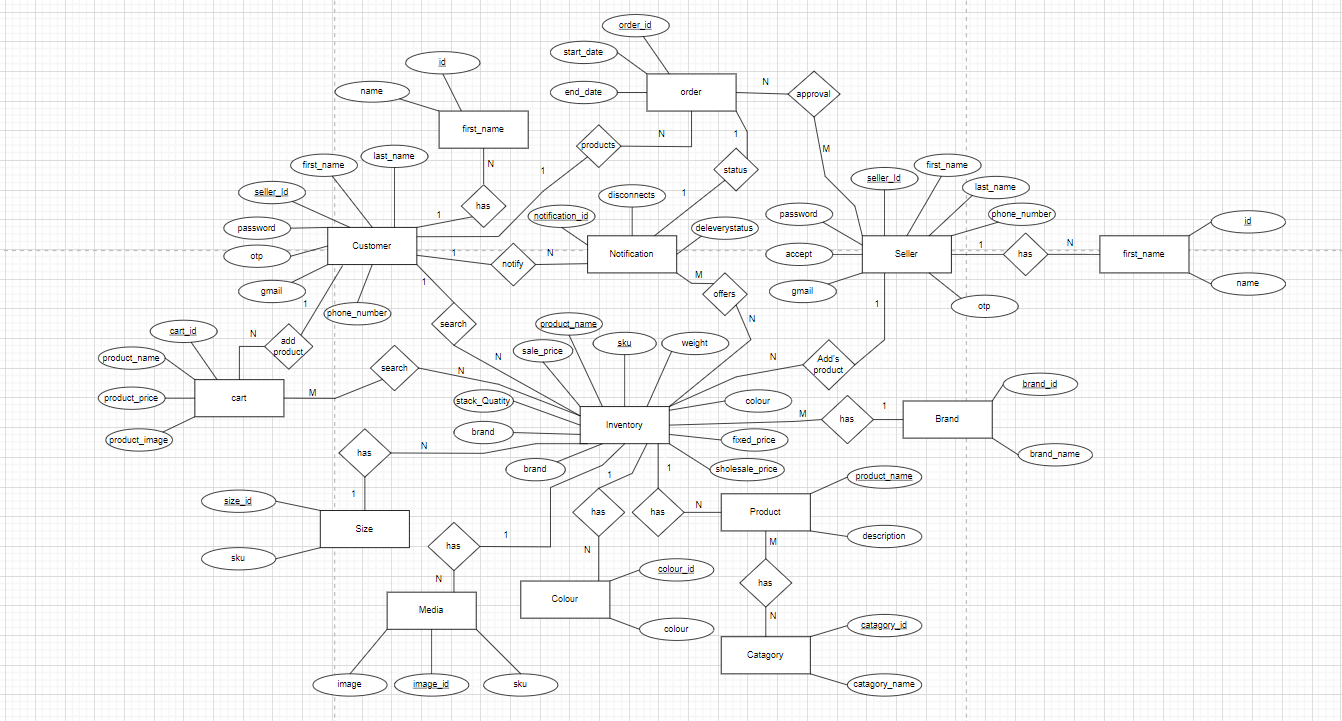
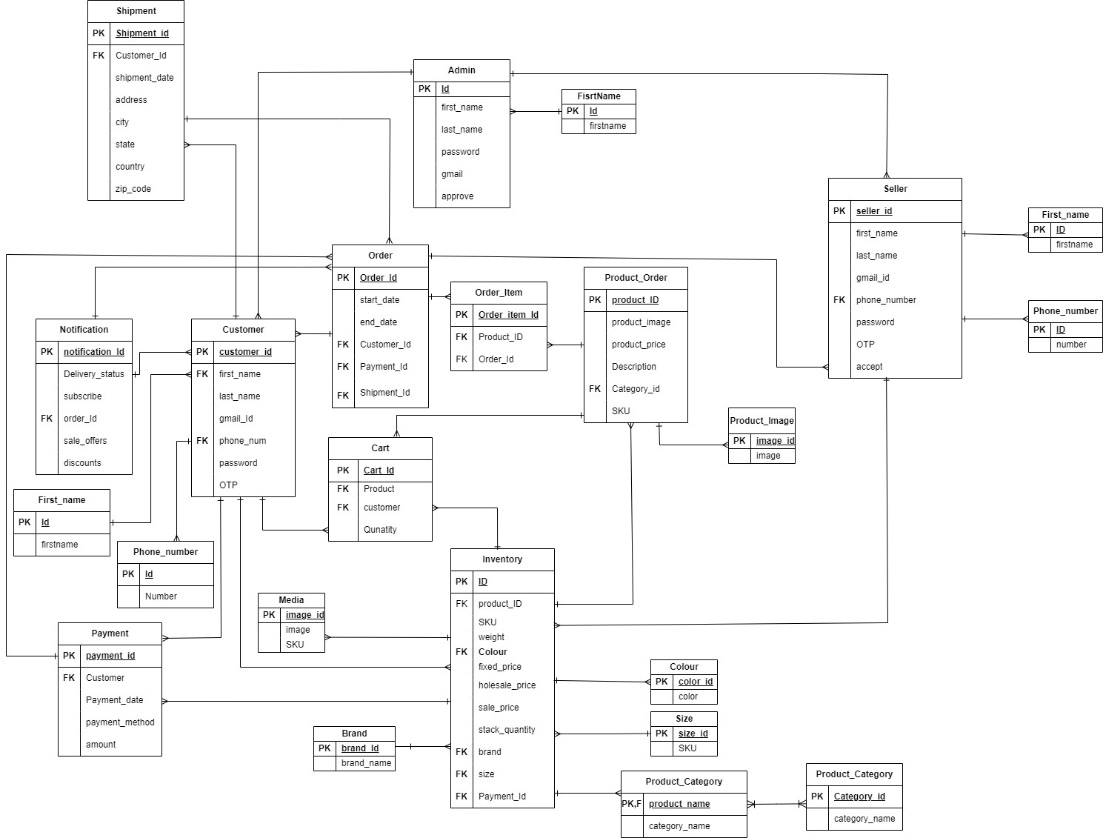
Database Design Assignment

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





* **Customer**: This entity represents the customers who create an account to place orders on the online shopping platform.
* **Product**: Represents the set of products available for purchase on the platform.
* **Product\_Category**: Categories in which the products are grouped.
* **Order**: Product orders placed by customers.
* **Order**\_**item**: Each item that is part of an order.
* **Payment**: The payment made by the customer once the order is completed.
* **Shipment**: Shipping information associated with an order, including delivery address and tracking information.
* **Cart**: The customer’s virtual basket or shopping cart, which stores items before they are purchased and become part of an order.
* **Seller**: This entity represents the seller can add the products and can accept order or reject order.
* **Inventory**: This entity represents that seller can add products, images and details of product.
* **Admin**: This entity represents approval of order for customer and seller.

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

Searching is essential for any e-commerce website because:

* It improves user experience by helping customers find products quickly.
* It boosts product discovery and increases sales.
* Searching saves time and makes shopping more efficient.
* Easy product finding leads to more purchases.
* Analysing search data provides insights into customer behaviour for better marketing strategies.

Here are a few ways to handle a searching and reporting:

* Replication and Load Balancing: Creating copies of the database (replicas) and directing read queries to them helps keep the system running smoothly, even when there's a lot of traffic.
* Indexing: Think of it as creating a roadmap for the database to find things faster. Indexes are like bookmarks that help retrieve data quickly, especially for common search terms like product names or categories.
* Denormalizing: Sometimes, duplicating data can speed up searches. It means storing information in more than one place to avoid complicated calculations when searching.
* Caching: Storing frequently searched results in memory using tools like Memcached or Redis. This reduces the need to repeatedly fetch data from the database, making responses faster.
* These methods work together to make sure that when customers search for products, they get results quickly, making the website more responsive and user-friendly.

This is one of the Technique -**Master-Slave Replication** involves having a main database (master) for writing data and one or more copies (slaves) for reading. The copies handle search and reporting queries, taking the load off the main database. Common tools for this include MySQL's replication, PostgreSQL's streaming replication, and Microsoft SQL Server's transactional replication.

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

* There are several major factors are considered for performance optimization:
* **Load Balancing**: In systems with multiple servers or nodes, load balancing ensures that the workload is distributed evenly, preventing individual components from overwhelmed and improving overall performance.
* **Database Indexing**: Indexing plays a crucial for database, where it allows use to retrieve the data faster for the database. Well-designed and properly utilized indexes can significantly improve query performance.
* **Regular Monitoring and Tuning**: Continuously monitor database performance and identify bottlenecks. Apply performance tuning techniques like query optimization and index optimization.
* Consider using **stored procedures** or prepared statements for frequently executed queries.
* **User authentication** is a limitation imposed by applications on users attempting to access them. Through authentication, an application can verify that each user attempting to log in is who he/she claims to be and has permission to access and use the application.
* **Scalability**: Ensure the database can handle increasing data and user traffic. Use horizontal scaling (adding more servers) or vertical scaling (increasing server resources).
* **Notifications**: we can use **triggers**, then all subscribed people can easily notify.

**4) Mention about Indexing, Normalization and Denormalization.**

Indexing: -

* SQL Index is a quick lookup table which is used to retrieve the data from the database very fast. It also retrieves a vast amount of data from the tables frequently.
* Index helps to speed up select queries but slows down insert and update queries.

Syntax: -

CREATE INDEX Index\_name ON Table\_name (column\_name)

The Index\_name is the name of the index that we want to specify.

The Table\_name is the name of the table that on which the index is to be created.

The Column\_name is used to specify on which column we need to specify the index.

* Example: Consider a database table storing customer information with columns like customer ID, name, email, and phone number. If the database frequently executes queries to search for customers by their email addresses, creating an index on the email column would significantly speed up these search operations.

Normalization: -

* The normalization is the process of organising the data in the database and it is a process of eliminate data redundancy and enhance data integrity in the table.
* Normalization is the process of divides larger table into the smaller tables and link them using relationships.

Steps to normalization: -

First Normalization Form(1Nf): -

* A relation is in first normal form if every attribute in that relation is singled valued attribute.

Table-1

|  |  |  |
| --- | --- | --- |
| seller\_Id | Seller\_Name | Phone\_Number |
| 1 | Ram | 9879879871,  9861515551 |
| 2 | Ramesh | 9898981234 |
| 3 | Suresh | 8787654321 |

After 1NF

|  |  |  |
| --- | --- | --- |
| Seller\_Id | Seller\_Name | Phone\_Number |
| 1 | Ram | 9879879871 |
| 1 | Ram | 9861515551 |
| 2 | Ramesh | 9898981234 |
| 3 | Suresh | 8787654321 |

Second Normalization Form(2Nf): -

* It needs to be in 1NF.
* All non-key attributes are fully functional dependent on the primary key. It does not have any non-prime attribute that is functionally dependent on any proper subset of any candidate key.

Three Normalization Form(3NF): -

* It needs to be in 2NF.
* No transition dependency exists.

Boyce Codd Normal Form (BCNF): -

* It needs to be in 3NF.
* Every Right-Hand Side (RHS) attribute of the functional dependencies should depend on the super key of that table.

Denormalization: -

* The denormalization it is the opposite to the normalization, where we can add redundant data to one or more tables one attribute can store multiply data.
* This helps us avoid costly joins in a relational database.

The ER diagram I will be used a normalization up to THE THIRD NORMAL FORM. The reason for this is that the online shopping system is purely transactional, so it must support constant and concurrent updates of the tables that make up the schema. And it must support those updates while strictly maintaining data integrity and consistency.

Indexing: Indexes are data structures that speed up data retrieval by allowing the database to quickly locate rows based on the values of certain columns. By creating indexes on columns frequently used in search conditions, the database can minimize the number of rows it needs to scan and thus improve search performance.

* Product, Product\_Category, Order, Order Item tables:

In these tables are followed normalization principles by representing separate entities and avoiding data redundancy. The Product table contains product-specific information, the Category table stores the different categories of products. And These tables represent order-related information. The Order table stores general order details such as order date, total amount, and references the Customer 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.

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

Scaling a database, it involves both vertical and horizontal scaling strategies to accommodate increased workload and user demand. Here's how I can handle the scaling:

* Vertical Scaling (Scaling Up):

Upgrade Hardware: Increase the resources (CPU, RAM, Storage) of the existing server to handle a larger workload. Optimize Configuration: Fine-tune database configurations, query optimizations, and caching mechanisms to maximize performance with existing resources. Vertical scaling is often limited by hardware constraints and may not be sufficient for handling significant increases in traffic.

* Horizontal Scaling (Scaling Out):
* Partition the database horizontally across multiple servers based on a shard key (e.g., customer ID, product category) to distribute the workload evenly.
* Set up read replicas of the database to offload read queries and distribute the read workload across multiple servers.
* Horizontal scaling it involves add multiply machines or nodes to distribute the workload, In this each machine work independently with their assigned work.
* The advantage of using horizontal scaling it can handle the large number of requests.
* By using the horizontal scaling improved the fault tolerance.
* As the workload grows, we can add the additional machines and increases the performance.
* Integrate queueing systems like RabbitMQ or Kafka to decouple and distribute background processing tasks such as order processing or data ingestion.
* Use distributed caching solutions like Redis or Memcached to cache frequently accessed data across multiple servers.
* Database-as-a-Service (DBaaS):
* **Utilize cloud-based database services like Amazon RDS, Google Cloud SQL, or Azure SQL Database**, which offer automated scaling capabilities. These services often provide features like auto-scaling, automated backups, and monitoring, making it easier to scale databases without manual intervention.

I have used the horizontal scaling. And by implementing a combination of these scaling strategies and continuously monitoring performance, the website can effectively handle increases in workload and user demand while maintaining optimal performance and reliability.

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

* The data types for attributes, such as uniqueness, nullability and referential integrity (foreign key relationships), are enforced at the database level to maintain consistency and integrity.
* The ER diagram assumes that data integrity constraints, such as referential integrity (foreign key relationships), are enforced at the database level to maintain consistency and integrity.
* To reduce data redundancy, I have used normalization and split the larger table into small tables and the columns which are stores multiply values divided them into the separate table by using foreign key.
* While ordering a product I have used two different tables in one table the product details will be saved and another table the order status will be stored like start date and end date because while ordering the data it may fail for some reasons which are exceptions for that the product should not get disturbed that's I was used two table
* By using triggers for notifications and discounts, sale offers.
* And by using stored procedures we can store the code and use it whenever possible.

These are some other points for implementation.

**Product Images** are stored directly as binary data (BLOB - Binary Large Objects) within the database tables.

* Instead of storing images directly, store URLs or file paths pointing to where the images are stored externally (e.g., on a server's filesystem or a cloud storage service).
* Only metadata such as image name, size, and format are stored in the database, reducing database size and improving performance.
* This approach requires managing image files separately from the database.

And coming to **authentication and authorization**, we cannot directly save the password, we can use hashing and adding with salt. The design of an authentication module needs to include a table that stores the information for validating each user’s login.

**Storing Password Hashes Instead of Passwords**

* Hashing functions, unlike encryption functions, are not bijective. It is practically impossible to obtain a password from its hash. However, hashing can be used to validate a password: if the password entered by a user renders the same hash as the one stored in the table, the hashing algorithm guarantees the password is the same as the one set by the user.

**Adding Salt to Passwords.**

* A salt is a random string generated when a password is set; it must be stored in the authentication data table along with the password hash. As a random factor but fixed for each user, it intervenes in the hashing function so that the password hash is unique even if a user chooses the same password as another user.
* Upon receiving the confirmation request from the user, the application extracts the token from the URL and looks for it in the user table. If it finds it in the data of the user account, it can guarantee that the email address associated with that account is valid.
* To add email confirmation to our schema, we need to add some fields to the login data table and a table containing the possible validation statuses. The confirmation token field must allow null values since it is cleared once the email address is confirmed.

**Password Recovery**

* This is done by requesting a user ID (usually a username or email address), validating that the user ID corresponds to an active user account, and sending the user an email with a randomly generated token to the previously validated email address.

And by using **Multi-Factor Authentication** – we can secure our website by hackers, and login safely.

And the complete schema, we can also include tables for assigning roles and permissions. These are the some of the assumptions for respective database design.