**Physical Entity Relationship diagram of database**

refer database.png

1. Seller can add multiple products, so there's a one-to-many relationship between Seller and Product.
2. Products can have multiple sku, so there’s a one-to-many relationship between Product and Product SKU
3. Each product can have multiple images, so there's a one-to-many relationship between Product and Image.
4. Buyers can add multiple products to their cart, and each product can be added to multiple carts, so there's a many-to-many relationship between Buyer and Product through Cart.
5. Buyers can have multiple orders, and each order can have multiple products, so there's a one-to-many relationship between Buyer and Order.
6. Sellers can have multiple orders, so there's a one-to-many relationship between Seller and Order.
7. Buyer can subscribe to multiple products and each product can be subscribed by many buyers, so there’s a many to many relationship between buyer and product through subscription
8. Buyers can receive multiple notifications, so there's a one-to-many relationship between Buyer and Notification.

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

For searching and reporting, we are going to use search engine like Apache Solr, it uses a master-slave architecture for achieving high availability, fault tolerance, load balancing, indexing. Solr can work as a NoSQL database with transactional support containing denormalized data so that we can have great searching performance.

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

Using search engine like solr to increase searching performance by using techniques:

1. Caching: it will store frequently accessed data to NoSQL storage so that its access speed can be vastly improved.
2. Load Balancing: it uses a master-slave architecture so it can create multiple slave nodes and then balance load to multiple nodes.
3. Denormalize: it stores data in denormalized form to avoid search query in complex join statements because it can make access time.

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

Indexing: Data that is not denormalized and is accessed frequently can be indexed to minimize the disc visits

Normalization: Database is normalized in our case to reduce redundancy and improve data integrity, data is broken down into simpler tables so that 1 piece of data is stored in only one place to avoid data inconsistencies.

Denormalization: Data retrieval is slow in normalized databases as query can have multiple joins to accommodate breaking of table during normalizing which reduces data retrieval speed therefore part of the data which is required frequently is store in denormalized so that data retrieval can be quick and easy, in our case to improve searching and reporting performance, we are storing product data in denormalized form so that buyers can traverse through products quickly.

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

We are using a normalized database due to which it is easier to add more features and functionality in our app, we can easily create new tables and link them with existing tables.