Database Design Assignment

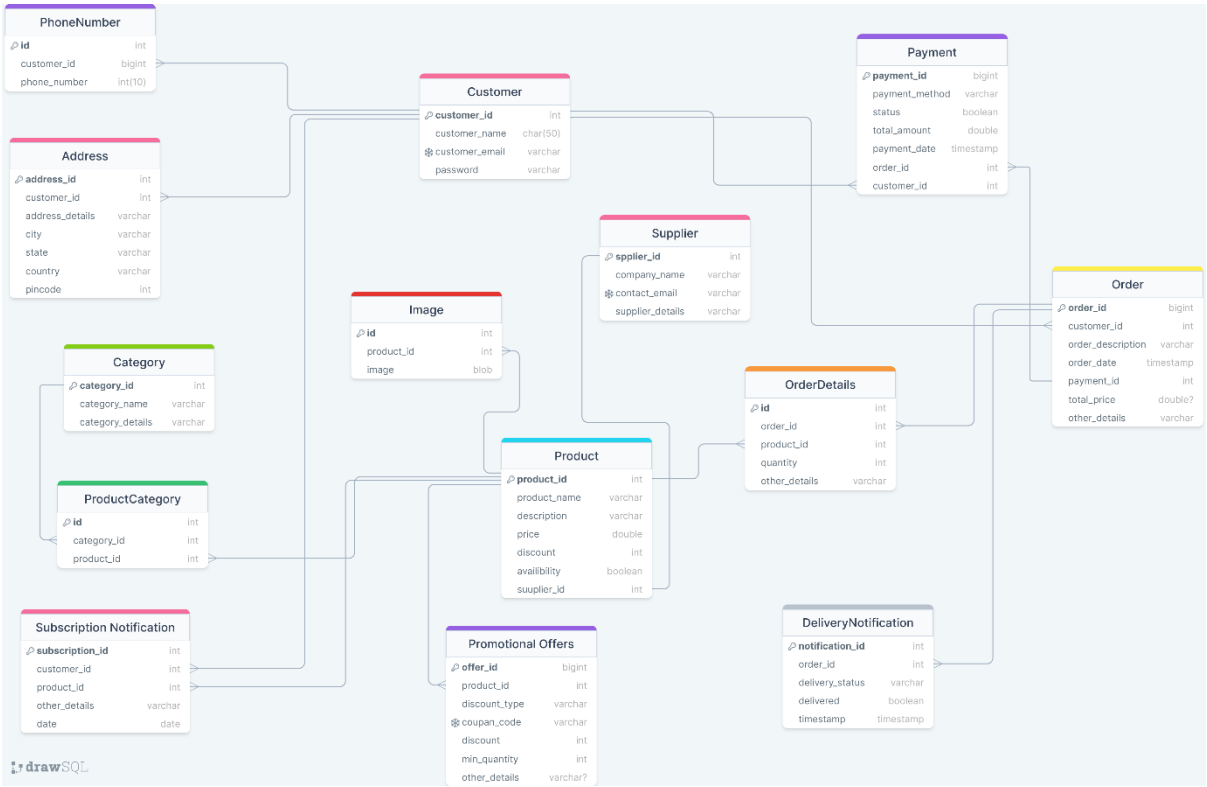
(E-Commerce)

Anurodh Singh

320

(Java Trainee)

1 . Entity Relationship Diagram:



For the different modules I have created the ER diagram. I have created entities by taking care of normalization concepts in mind. As there are some attributes namely “phoneNumber” and “addresses” may have multiple values, which is being normalized by dividing into two tables. As first normal says there should not be any attributes in table which is multivalued.

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

There must be some point we need to take into account in order to get the higher searching performance.

1. **Indexing**: Proper indexing of columns used in search queries can significantly improve search performance. Identify the columns frequently used in search conditions i.e product_name, category etc. and then create the appropriate indexes for them.

2. **Query Optimization**: We need to write the efficient SQL queries by optimizing the joins, filter, and sorting techniques. By use of EXPLAIN or similar tools to analyze query execution plans and identify areas of optimization.

3. **Searching Tools:** For the complex search requirements, involving text matching, consider using full text-search capabilities provided by some databases. As use of ElasticSearch for the advanced text-search capabilities.

For the replication SQL for searching and reporting:

1. **Replication Types**: Depending on our requirements, choose an appropriate replication type. My SQL for example supports various replication types like statement-based, row-based or mixed based replication each with its own trade-offs in terms of performance and data consistency.
2. **Read replicas:** Consider using read replicas for offloading read heavy operations such as searching and reporting. Read replicas can improve search performance by distributing read queries pass multiple replicas reducing the load on primary database.
3. **Load balancing:** Use low balancing techniques to distribute search and reporting queries evenly across available replicas optimising resource utilization and query response time.
4. **High Availability and failover:** implement high availability strategies to ensure continuous availability of search and reporting functionality in case of primary database failures.

3. Major Factors taken into consideration for performance:

1. **Database Schema Design:** A well designed database schema can significantly impact performance. Normalize the schema to reduce redundancy and improve data

integrity. However, denormalization might be considered for performance critical queries to reduce join operations.

2. **Indexing**: Proper indexing of columns used in search, filter and join operations can greatly improve query performance. Identify frequently used columns and create indexes based on the query patterns.
3. **Normalization and query optimization**: Normalized table and optimized query is crucial for performance.
4. **Database Engine**: Choose a database engine that aligns with application requirements. Different database engines have different strengths and weaknesses, so selecting the right one can impact performance significantly.
5. **Caching and connection pooling**: Implement caching mechanisms to store frequently accessed data in memory, reducing the need for repeated database queries. Manging the database connections effectively is important for performance.

4. Mention about Indexing, Normalization and Denormalization.

Indexing: A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records. Identifying the frequently queried fields such as product name, category and SKU. Creating the indexes on these fields to speed up the search queries.

Normalization: Normalization is the process of organizing data in a database. It includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency. Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A customer address change is easier to implement if that data is stored only in the Customers table and nowhere else in the database.

Denormalization: Denormalization is a strategy used on a previously normalized database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data or by grouping data.

5. Handle the scaling if required at any point of time.

There is various scaling handling techniques are there in order to scale up the requirement at any point of time. By using the different cloud services now a days made this very easy to scale up and down the requirements as per the need. There are two ways we can scale up the application.

Vertical Scaling: We can increase the capacity of existing server e.g CPU, memory, storage to handle the increased load.

Horizontal Scaling: We can add more servers to distribute the load. This can involve sharding the database multiple servers or using the distributed database system.

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

I have assumed that the single database is being used for the application. The database may be MySQL or PostgreSQL in order to fulfilment of all the desired requirements. As in order to complete the requirement of concurrently handle the 100k users at same time, we can have the approaches of database optimization, indexing, scaling up, load balancers and database replication. As here in the application many transactions-based queries are there which may involves the two or more entities in order to make the transaction-based insertion.
As a example, for the Order created on application involves managing the transactional data related to customer orders, items and customer information. Like if order is created it should be atomic, for the same purpose the product availability and user payment should be successfully handled simultaneously. For such atomic transactions (ACID property) we need the databases like MySQL or PostgreSQL for their support for transactions, relational data modelling and scalability.

As for some modules it may that we can use the different database, such as NoSQL for some reporting-based work or searching specific database, but it would be difficult for the application to handle the document based and relation-based database simultaneously. NoSQL databases could be used in inventory modules as searching would be efficient in case of document based storage.

In order to achieve the microservice specific structure, it may be that we will be using the different database for the different modules. i.e Database-per-service architecture.
But the problem with this approach is ACID property and transaction-based data insertion, updation and deletion. Somehow, there are solutions for the database-per-service architecture to handle the modularity of transactions, that is we may use API for each service and handle the transaction.

References:

https://drawsql.app/
https://microservices.io/
https://microservices.io/patterns/data/database-per-service.html
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html