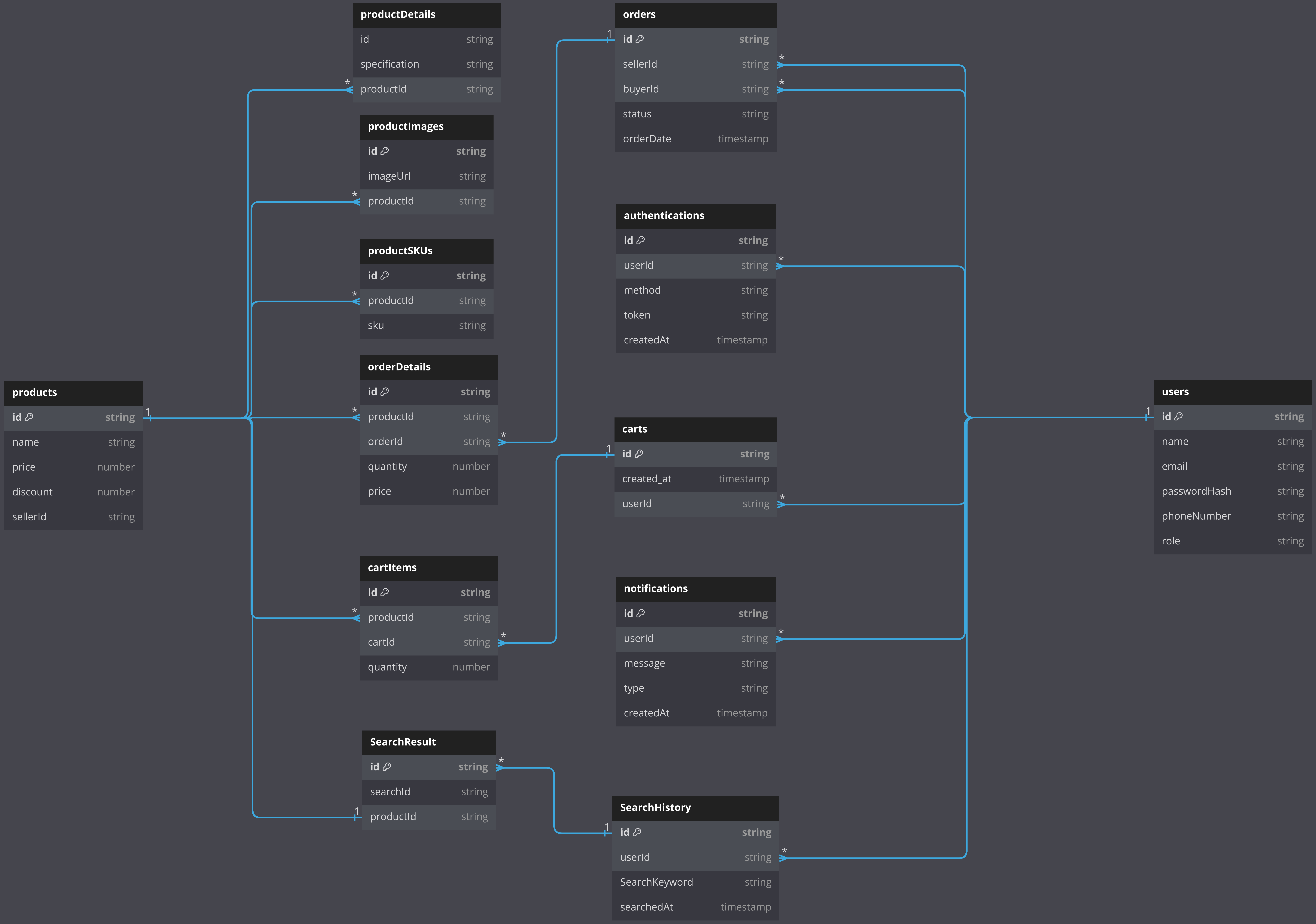

productDetails	
id	string
specification	string
productId	string

productImages	
id	string
imageUrl	string
productId	string

productSKUs	
id	string
productId	string
sku	string

orderDetails	
id	string
productId	string
orderId	string
quantity	number
price	number

cartItems	
id	string
productId	string
cartId	string
quantity	number

SearchResult	
id	string
searchId	string
productId	string

products	
id	string
name	string
price	number
discount	number
sellerId	string

orders	
id	string
sellerId	string
buyerId	string
status	string
orderDate	timestamp

authentications	
id	string
userId	string
method	string
token	string
createdAt	timestamp

carts	
id	string
created_at	timestamp
userId	string

notifications	
id	string
userId	string
message	string
type	string
createdAt	timestamp

SearchHistory	
id	string
userId	string
SearchKeyword	string
searchedAt	timestamp

users	
id	string
name	string
email	string
passwordHash	string
phoneNumber	string
role	string

dbdiagram.io

Database Design Assignment QnA

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

Ans: In the context of databases, specifically SQL databases, "searching performance" refers to how quickly and effectively queries may be run to obtain pertinent data. Managing replication is one component that can affect reporting capabilities as well as searching efficiency, among other aspects.

Performance of Searches:

1. **Indexes:** Search efficiency may be greatly improved by appropriately indexing the columns utilized in the search criteria. The database engine can find and retrieve data more rapidly with the aid of indexes.
2. **Query Optimization:** Effective searching depends on well-crafted SQL queries. This entails designing queries to maximize index utilization and reduce pointless calculations.
3. **Normalization and denormalization:** These are two different processes that include structuring data to minimize redundancy and adding redundancy for performance improvements, respectively. Your database schema may be normalized or denormalized, depending on the particular use case.
4. **Partitioning:** Partitioning is the process of breaking up huge tables into more manageable sections. By enabling the database engine to concentrate on pertinent partitions when running queries, this can enhance searching performance.
5. **Caching:** By using caching techniques, you may lessen the number of times you have to run the same queries. Response times can be speed up by serving caching results for data that is often queried.

Managing Replicas for Reporting & Searching:

The process of making and keeping copies of a database in several places is known as database replication. High availability, load balancing, and fault tolerance are common uses for replication. The following describes how to manage replication for reporting and searching:

1. **Create read-only copies** of your database to delegate read-intensive tasks like reporting and searching. By doing this, the burden is distributed, and system performance is enhanced overall.
2. **Load Balancing**: To divide up queries among several replicas, use load balancers. By doing this, the responsibility of reporting and searching is divided equally, and replicas are kept from becoming bottlenecks.
3. **Consistency Models:** Select a consistency model that meets the needs of your application. Certain programs possess the ability to endure eventual consistency, hence providing more adaptability in allocating read operations among replicas.
4. **Real-time Replication:** To make sure the replicas are current, use real-time or almost real-time replication. For reporting reasons, this is essential since reports with outdated data may be erroneous.

5. **Establish a failover system** to deal with the eventuality that one of the replicas stops working. This guarantees that the database will always be accessible for reporting and searching.
6. **Index Maintenance:** To guarantee that searching speed is constant, maintain consistent indexes across copies. To prevent performance loss, changes to the indexing scheme should be carefully handled.
7. **Scaling and Monitoring:** Keep track on replica performance and adjust our infrastructure as necessary. As our application's demands change, adjust the number of copies.

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

Ans: The performance of databases is essential for any application that uses them. Users may become irritate, lose productivity, and possibly see a decline in sales due to a sluggish database. Understanding the key elements that affect database speed is crucial to improving database and guaranteeing its smooth functioning.

1. **Workload:**
   - Performance is greatly impacted by the kind and quantity of operations that are requested from the database.
   - Large data quantities, intricate queries, and frequent read/write cycles can put a load on the system.
   - Anticipating resource requirements and optimizing appropriately may be achieved by examining the average workload of your program and pinpointing periods of high usage.
2. **Resources for Hardware and Software:**
   - The hardware that powers databases for examples – CPU, RAM, etc is essential for performance.
   - Processing may become sluggish and create bottlenecks due to a lack of resources.
   - Performance may be greatly enhanced by selecting hardware that is suitable for your task and making sure that there is enough RAM allocated for caching frequently requested data.
3. **Design of Database Schemas:**
   - The query performance can be greatly impacted by the way data is stored and arranged inside the database structure.
   - Selecting the appropriate data formats, building suitable indexes for commonly used columns, and normalizing tables to reduce redundancy can all help to speed up data retrieval and processing.

4. **Optimization of Queries:**
   - Queries that are poorly crafted can waste resources and cause database operations to lag.
   - Query efficiency may be greatly improved by comprehending query execution plans, utilizing appropriate joins, and indexing techniques, and minimizing pointless data fetching.

5. **Control of Concurrency:**
    - Controlling concurrent access to data in multi-user scenarios is essential to preserving data integrity and averting performance problems.
    - Data consistency is ensured and conflicts between concurrent processes are avoided by locking mechanisms and transaction management strategies.
6. **Tuning and maintaining databases:**
    - To sustain optimal performance, bottlenecks should be found, tuning strategies should be put in place, and database performance should be routinely monitored.
    - This might entail modifying setup options, maintaining indexes, and rearranging databases.

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

Three Crucial Methods for Enhancing Database Performance: Indexing, Normalization, and Denormalization.

Each of these three tactics, which have advantages and disadvantages of their own, is essential for maximizing database performance:

1. Indexing

Consider a book where the back has an index. You may swiftly locate information using it without having to read the full book. Like this, database indexing generates a supplementary data structure that associates certain values with the relevant table data places.

Benefits:

- Quicker data retrieval for search criteria and columns that are used often.
- enhances the speed of queries that use sorting and joins.
- reduces the number of database scans, therefore CPU utilization.

Cons:

- The expense of building and updating indexes.
- Indexes require more storage capacity.
- Not every query may profit from it equally.

2. Normalization

- The goal of normalization is to reduce data abnormalities and redundancies in the structure of the data. This entails dividing the data according to their logical dependencies and linkages into distinct tables.
- As data organization and integrity levels increase, several normal forms (1NF, 2NF, and 3NF) characterize them.

Benefits:

- Saves storage space and lessens data redundancy.

- Improves data consistency and integrity.
- simplifies updating and changing data.

Cons:

- To extract similar data, more intricate joins could be needed.
- may cause some queries to run more slowly because of more joins.

3. **Denormalization:**
    - Denormalization purposefully adds some redundancy to boost query efficiency. To save money on joins and data retrieving from several tables, it purposefully duplicates data in some tables.
    - This procedure is often done after normalization has been achieved.

Advantages:

- much quicker query execution for complicated queries that are used often.
- enhances efficiency in applications that need a lot of reading.
- reduces query complexity by grouping related data.

Cons:

- raises the need for storage by increasing data redundancy.
- if changes are not handled correctly, introduces possible concerns of data inconsistency.
- may make updating and maintaining data more difficult.

Selecting the Appropriate Strategy:

The best course of action is determined by your unique requirements and priorities. In general:

Normalization: Data integrity is vital, complicated joins are uncommon, and it is preferred for applications that require a lot of writing.

Denormalization: Perfect for applications with a lot of reading, where joins are common and query efficiency is crucial.

Hybrid Approach: A useful way to strike a balance between data integrity and performance is to combine components that are normalized and denormalized.

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

Ans: Increasing a database's capability to manage an increasing demand, a growing volume of data, or both is known as scaling. There are several approaches to database scalability; the one we use will rely on the needs and features of our application. These are a few typical scaling techniques:

1. **Vertical Scaling:**
   - Boost a single machine's capacity by increasing its RAM, CPU, or storage.
   - Suitable for databases that can benefit from a bigger, more powerful server.
   - restricted by a single machine's maximum capacity.
2. **Horizontal scaling:**
   - Divide up the database over several computers.
   - Each machine manages a portion of the workload or data.
   - allows for greater scalability because additional computers may be added as needed.
   - Common in NoSQL databases and distributed systems.
3. **Sharding:**
   - Divide the database into shards—smaller, easier-to-manage sections.
   - Every shard is kept on a different server.
   - Effective for both reading and writing tasks, but difficult to set up and maintain.
4. **Replicate:**
   - Make duplicates, or copies, of the database on other servers.
   - Performance may be increased by distributing read requests using read replicas.
   - Replication can boost fault tolerance and availability.
5. **Caching:**
   - To keep data that is often retrieved in memory, use caching methods.
   - serves certain queries from the cache, which lessens the strain on the database.
   - utilized often for tasks involving a lot of reading.
6. **Load Balancing:**
   - Ask incoming database requests to be split among many servers.
   - makes certain that no server acts as a bottleneck.
   - Commonly used in combination with replication or sharding.
7. **Cloud-based Solutions:**
   - Make use of cloud-based database services with infrastructure that is scalable.
   - Managed database systems with automated scalability are frequently offered by cloud providers.
8. **Enhancing Indexing and Queries:**
   - Boost the effectiveness of database operations and queries.
   - Effective query optimization and indexing may drastically lower the database's load.

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

Ans: When designing the Entity-Relationship (ER) diagram for this e-commerce website with the mentioned modules and features, several assumptions I made to create a comprehensive and effective representation of the system. Here are some assumptions I considered:

User Roles:

- I Assumed that there are two main user roles: sellers and buyers.
- Customers may browse products, put items in their carts, and submit orders.
- Sellers can take orders, add items, and manage them.

Product Illustration:

- Numerous properties, such as name, description, price, discount, specification, and SKU, are possible for products.
- I Assumed that there is a one-to-many link (a seller may have more than one product) between them.

Module for Inventory:

- Let's say that a product can have more than one picture.
- A product's SKU (stock keeping unit) allows you to identify any variants that may exist.
- Sellers can add, update, and remove goods from their inventory.

Cart/Order Module:

- Using the shopping cart, assume a many-to-many link between customers and items.
- Order information includes order items (things in the order), order date, and order status.
- Orders may be accepted or rejected by sellers.

Module of Notification:

- Assume that customers may sign up to get alerts when certain goods become available.
- Buyers receive notifications when an order status changes (e.g., packed, shipped, ready to ship).
- Push notification support for special offers and discounts during sales.

Authentication & Authorization:

- Sellers and buyers can log in using a phone number and an OTP or with an email address and password.
- Support for the "Forget Password" option for both buyers and sellers.

Scalability and Concurrency:

- Assume that 100,000 active users may all look for items at once in a concurrent environment.
- Considered about how the design must be scalable to accommodate possible future expansion.

Security Procedures:

- Put in place the appropriate security measures to safeguard private data, particularly during the authentication process.
- For sensitive data, thought about encryption.

Normalization:

- To cut down on duplication and guarantee data consistency, normalize the database structure.
- To create the best possible design, determine the functional linkages and dependencies between elements.

Database Options:

- Be aware that depending on their requirements, separate modules can need different databases.
- Thought about NoSQL databases for flexible data models and relational databases for structured data.