**Database Design Assignment**

**E- Commerce Application**

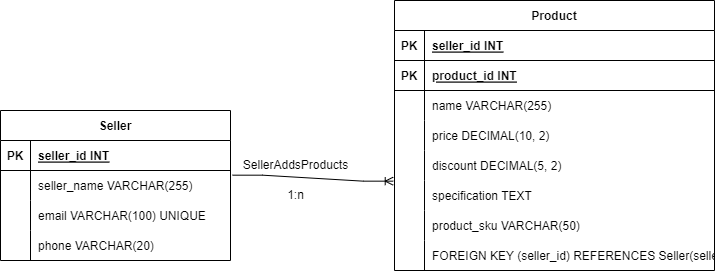
**For this assignment, I have opted to implement a microservice architecture, wherein each module functions as an independent microservice, promoting modularity, scalability, and maintainability.**

**Inventory Module :**

**Entities: Product, Seller**

**Attributes for Product: Product ID (Primary Key), Name, Price, Discount, Specification, Product SKUs, etc.**

**Relationships: Seller can add products (One-to-Many relationship: Seller to Product)**

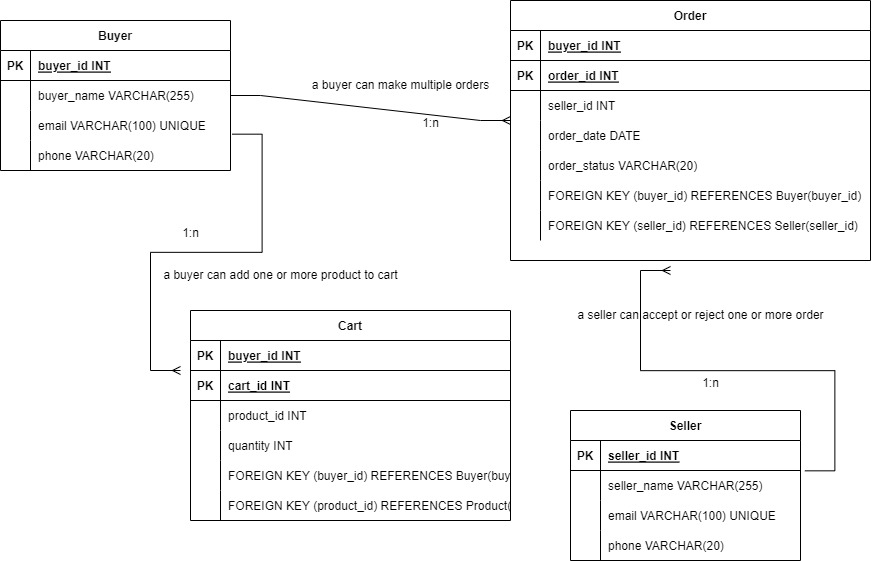
****

**Order/Cart Module:**

**Entities: Buyer, Order, Cart**

**Attributes for Order: Order ID (Primary Key), Order Date, Order Status, etc.**

**Relationships: Buyer can have multiple orders (One-to-Many relationship: Buyer to Order), Buyer can have a cart with multiple products (One-to-Many relationship: Buyer to Cart), Seller can accept or reject orders (One-to-Many relationship: Seller to Order)**

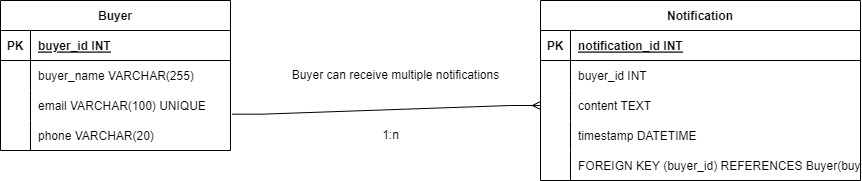
****

**Notification Module:**

**Entities: Buyer, Notification**

**Attributes for Notification: Notification ID (Primary Key), Content, Timestamp, etc.**

**Relationships: Buyer can receive multiple notifications (One-to-Many relationship: Buyer to Notification)**

****

**Authentication and Authorization:**

**Entities: Buyer, Seller**

**Attributes for Buyer/Seller: User ID (Primary Key), Email, Password, Phone number, etc.**

**A screenshot of a computer

Description automatically generated**

**Searching Performance & Replication in SQL for Searching & Reporting:**

To improve searching performance, you can create indexes on frequently searched columns, such as product names, product IDs, and other attributes used for filtering. This will help in faster data retrieval during searches. Additionally, you can use caching mechanisms to store frequently accessed data in memory to reduce the load on the database.

For replication in SQL, you can use database replication techniques like master-slave replication or multi-master replication. Replication helps distribute the read load among multiple database instances, thereby improving read performance. For reporting purposes, you can set up separate read-only replicas or use materialized views to store aggregated data for reporting queries, reducing the load on the main database.

**Some major factors to consider for performance are:**

* Proper indexing of frequently used columns.
* Efficient database design, including normalization to avoid data redundancy.
* Caching mechanisms to reduce database queries.
* Optimal query design and usage of appropriate joins.
* Hardware resources and server capacity.
* Load balancing to distribute traffic evenly.
* Query optimization and tuning.
* Efficient use of memory and disk space.

**Indexing, Normalization, and Denormalization:**

* **Indexing:** Indexing is a technique used to speed up data retrieval operations in a database. It creates a separate data structure (index) that contains the keys and pointers to the actual data. Indexes allow the database to quickly find rows based on the indexed columns, improving search performance.
* **Normalization:**  Normalization is the process of organizing data in a database to eliminate redundancy and improve data integrity. It involves dividing a database into tables and defining relationships between them. The main goal of normalization is to minimize data duplication and prevent update anomalies.
* **Denormalization:** Denormalization is the process of deliberately introducing redundancy into a database by combining tables to improve query performance. It can be useful in read-heavy systems to reduce the number of joins required for complex queries. However, denormalization should be used carefully, as it can lead to data integrity issues and increased storage requirements.

**Handling Scaling:**

To handle scaling, you can employ various strategies such as

* **Vertical scaling:** Upgrading hardware resources (CPU, RAM, etc.) of the existing server to handle increased load.
* **Horizontal scaling:** Distributing the load across multiple servers by adding more nodes to the system.
* **Sharding:** Partitioning data across multiple databases or servers based on certain criteria (e.g., user ID, region).
* **Load balancing:** Distributing incoming traffic among multiple servers to avoid overloading a single server.
* **Caching**: Using caching mechanisms to reduce database load and improve response times.
* **Cloud-based solutions:** Utilizing cloud services that provide scalable infrastructure on-demand.

**Assumptions:**

* The application is primarily web-based and accessible to users through browsers and/or mobile devices.
* The system will use a relational database (SQL) to store data.
* The number of active users searching for products (100K) is considered the peak concurrent load on the system.
* Users are primarily identified by their email, password, and phone number for authentication.
* The application will support push notifications using appropriate notification services.
* The system will have a centralized authentication and authorization mechanism.
* The seller will have the authority to manage their inventory and process orders.
* The ER diagram is a simplified representation and may not include all attributes and relationships.

**Handling 100K active users searching for a product concurrently is a significant technical challenge that requires careful design and implementation. Several technological strategies can be employed to meet this requirement:**

* **Scalable Infrastructure:** To handle a large number of concurrent users, the application should be deployed on a scalable infrastructure. Cloud-based solutions like AWS, Google Cloud, or Microsoft Azure can provide auto-scaling capabilities, allowing the application to automatically add more resources (such as servers) to handle increased user demand during peak times.
* **Load Balancing:** Implement load balancing to distribute incoming requests across multiple application servers. This ensures that the load is evenly distributed, preventing any single server from becoming a bottleneck.
* **Caching:** Utilize caching mechanisms to store frequently accessed data in memory. This can significantly reduce the number of database queries and improve response times, especially for read-heavy operations like searching for products.
* **Database Optimization:** Optimize the database design and indexing to handle high traffic efficiently. Use read replicas or distributed databases to distribute the read load across multiple database instances.
* **Asynchronous Processing:** For tasks that do not need to be completed immediately, consider using asynchronous processing and message queues. For example, sending notifications can be queued and processed in the background, reducing the response time for the main user-facing operations.
* **Content Delivery Network (CDN):** Use a CDN to cache and deliver static assets like product images to users from servers located closer to their geographical locations. This reduces the latency and improves the loading speed of product images.
* **Microservices Architecture:** Consider breaking down the application into smaller, independent microservices. This allows for better scalability, as each microservice can be scaled independently based on its specific demand.
* **Optimized Front-end**: Optimize the front-end code and assets to ensure fast loading times and reduce the strain on the servers.
* **Asynchronous Search**: Consider implementing an asynchronous search mechanism where search results are returned incrementally as they are processed. This can help maintain responsiveness during high search loads.
* **Circuit Breaker Pattern:** Implement the circuit breaker pattern to prevent cascading failures and provide graceful degradation during spikes in traffic.
* **Monitoring and Load Testing:** Regularly monitor the application's performance and conduct load testing to identify bottlenecks and performance issues. This allows for proactive optimization and capacity planning.
* **Geographical Distribution:** If the application serves users from different regions, consider deploying servers in different geographical locations to reduce latency and improve the user experience.

Overall, a combination of scalable infrastructure, caching, database optimization, asynchronous processing, and load balancing techniques will help ensure that the application can handle the high demand of 100K active users searching for products concurrently. Continuous monitoring, tuning, and testing are essential to maintain optimal performance as user demand grows over time.