# Ecommerce application database schema documentation

## Table of contents:

1. Introduction
2. Database table entries
3. Assumptions
4. Searching performance Improvement
5. Major factors are taken into consideration for performance and scaling (if required).
6. Indexing, Normalization and Denormalization

## Introduction:

This document provides an overview of the database schema for the E-Commerce App. The schema defines the structure of the database, including entities, relationships, and attributes. The app facilitates various functionalities such as inventory management, order processing, notifications, and user authentication.

Also, this documentation defines scaling and searching like scenarios for better performance of the application.

## Database table entries:

* User: Represents a user of the application, either a buyer or a seller.
* Products: Represents a product available for purchase
* Product Image: Represents images associated with a product.
* SKU: Represents stock-keeping units for products.
* Cart: Stores items added to a user's shopping cart.
* Order: Represents an order placed by a user.
* Order Item: Represents individual items within an order.
* Product Notification: Stores notifications related to product availability.
* Order Notification: Stores notifications related to order status.
* Push Notification: Stores notifications for push alerts to users.
* Address: Stores addresses of the user.
* Roles: Stores user role. E.g., Buyer and seller.

## Assumptions:

* User can login via email and password
* To reset password, an email with reset password link will be sent to user’s email.
* User’s will have two roles buyer and seller and stored to database respectively.
  + “1” for buyer
  + “2” for seller
* User can search only for the products by “product\_name”.
* Assume we have used AWS cloud services.

## Searching & It’s performance improvement:

In this application we’ll be using Elasticsearch. Elasticsearch is a powerful, distributed search and analytics engine that's well-suited for text-based searches.

1. Install and Set Up Elasticsearch:

* We used Elasticsearch on a AWS cloud hosted Elasticsearch service.
* Configure Elasticsearch settings, including index configuration, analyzers, and mappings.

2. Indexing Data:

* Set “product\_name” as an index (else we can also use “product specifications”).
* Create an index in Elasticsearch for your products.
* Map product schema to Elasticsearch's document structure.

3. Integration with Application:

* We need to use an Elasticsearch client library (Depending upon the application backend language) in the application’s backend to interact with the Elasticsearch cloud cluster.

4. Indexing Products:

* Every time there is any update in product table (like: new product, delete product), need to update corresponding Elasticsearch index.

5. Search Queries:

* + - When a search happens, need to construct a query using Elasticsearch’s Query DSL and sent this query to the Elasticsearch using client library.

6. Search Results:

* + - Receive search results from Elasticsearch.
    - Present these results to users in your application's frontend, using your preferred UI.
* Also, we can use Pagination, sorting, filtering, auto-complete etc. feature using Elasticsearch’s features.

### Performance improvements:

Improving search functionality in this e-commerce application involves optimizing the data and enhancing the user experience. Here are some strategies we can consider:

1. Indexing Strategy:

* Experiment with different indexing techniques to improve the quality of search results.

2. Query Construction:

* Use a combination of different query types, such as match queries, multi-match queries, and fuzzy queries, to capture variations in user input.
* Explore more advanced queries (like prefix queries etc.) for specific use cases.

3. Auto-Suggest and Auto-Complete:

* + - Enhance auto-suggest and auto-complete functionality to guide users and improve query accuracy.

4. Caching and Pagination:

* Implement caching for frequently searched terms and popular products to reduce query load.
* Efficiently implement pagination to avoid overwhelming users with too many results at once.

5. Performance Optimization:

* Optimize the performance of your Elasticsearch cluster, including replicas, and hardware resources.
* Consider using Elasticsearch's search profiling to analyze query execution.

6. Regular Re-indexing:

* Periodically refresh or re-index data to incorporate updates and changes.

## Factors taken into consideration for performance & scaling the application:

Handling search traffic from 100K users concurrently requires careful planning and optimization. Here are some strategies to ensure this e-commerce application can handle such a load:

1. Distributed Architecture:

* We need to deploy Elasticsearch in a distributed cluster to distribute the search load across multiple nodes. This improves scalability and redundancy.

2. Replication:

* We need to configure replicas to provide fault tolerance and distribute read traffic.

3. Hardware Resources:

* We need to use high-performance hardware and AWS cloud instances to host your Elasticsearch nodes.
* We need to use vertical scaling (increasing instance size) and horizontal scaling (adding more instances) as needed.

4. Load Balancing:

* We need to use load balancers in front of Elasticsearch cluster to evenly distribute search requests.

5. Auto-Scaling and Elasticity:

* We need to use AWS cloud services to automatically scale the Elasticsearch cluster up or down based on demand.

### Scaling the application:

1. Vertical Scaling:

* Upgrade the hardware resources of Elasticsearch nodes, such as CPU, memory, and storage, to handle increased load.

2. Horizontal Scaling:

* Add more Elasticsearch nodes to the cluster to distribute the load and increase capacity.
* Elasticsearch's distributed architecture allows us to scale horizontally by adding more nodes as needed.

3. Load Balancing:

* Implement a load balancer in front of the Elasticsearch cluster to distribute incoming search requests among the nodes.
* This helps ensure even distribution of traffic and prevents any single node from being overwhelmed.

5. Cloud Services:

* AWS has managed Elasticsearch services, and this service automated the scaling, monitoring and maintenance features. We can use that here.

6. Planning for Peaks:

* We can plan traffic spikes during events, or holidays and scale the application accordingly.

## Indexing, Normalization & Denormalization:

1. Indexing:

Indexing involves creating data structures that enhance the speed of data retrieval operations, such as searching or filtering. In the context of this e-commerce application, efficient indexing is essential for providing fast search results to users.

* Used in Search: In this Elasticsearch-based search system, indexing refers to the process of storing and organizing data in a way that allows for fast and accurate search queries. Here we can use products for indexing.

2. Normalization:

Normalization is a database design technique that minimizes data redundancy by breaking data into smaller, related tables to maintain data integrity and reduce update anomalies.

* Usage in this E-Commerce:
  + In this relational database, we can use normalization to efficiently manage user data, product details, and order information. For example, we have separate tables for Users, Products, Orders, and Order\_Items. This reduces redundancy and ensures that each piece of information is stored in one place, preventing inconsistencies.

3. Denormalization:

Denormalization involves combining related data into a single table to improve query performance. While it increases redundancy, it can speed up certain types of queries.

* Usage in Search and Analytics:
  + In Elasticsearch indices, we might de-normalize data by including related information within each document. For example, we could include product details within the Order\_Items document to avoid additional lookups when displaying order details. This will speeds up query execution at the cost of increased storage.