E-Commerce Database Management System Design Report

Project Group 14 – IS 455 Database Design & Prototyping
Group Members: Bharath Ganesh, Juhi Khare, Alisha Rawat

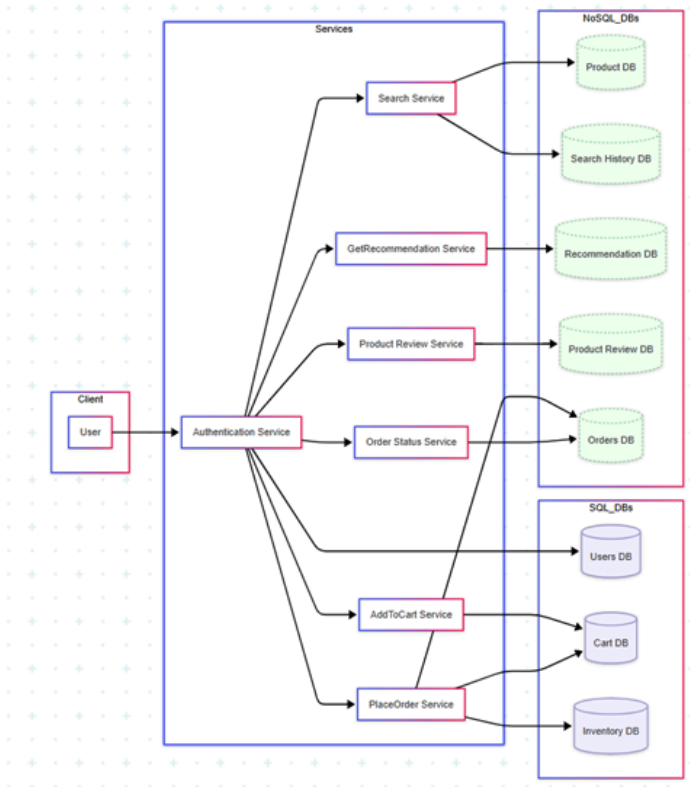
IS 455
Prof. Yang Zhang
May 15th, 2025
University of Illinois Urbana Champaign

1. Problem Definition

Our objective is to design a robust, scalable database system for an E-Commerce platform that ensures seamless functionality for millions of users. The primary system capabilities include:

- **User Account & Profile Management**: Secure sign-up, login, and profile maintenance.
- **Product Browsing & Search**: Efficient catalog navigation with filters and intelligent search suggestions.
- **Cart & Checkout**: Smooth user experience for adding to cart, processing payments, and confirming orders.
- **Order Tracking**: Real-time order status and updates.
- **Feedback & Recommendations**: Personalized recommendations, user reviews, and ratings.

2. System Architecture Overview



The following diagram illustrates the overall architecture of the E-Commerce system, demonstrating the interaction between the user, core services, and both SQL and NoSQL databases. It effectively captures how different system components are orchestrated to meet the functional requirements outlined in the problem definition.

Explanation of Components

- **Client (User)**
  The end user interacts with the system via a frontend client (web or mobile app). All requests go through the **Authentication Service** for access control.

- **Authentication Service**
  This service validates users using OAuth2/JWT-based mechanisms and routes their requests to appropriate internal services based on role and privileges.

- **Service Layer**
  Each backend service handles a specific set of operations:

    - **Search Service**: Interfaces with the **Product DB** and **Search History DB** to fetch products and log searches.

    - **GetRecommendation Service**: Retrieves personalized recommendations from the **Recommendation DB**.

    - **Product Review Service**: Manages user reviews and ratings using the **Product Review DB**.

    - **Order Status Service**: Fetches real-time order details from the **Orders DB**.

    - **AddToCart Service**: Updates cart information in the **Cart DB**.

    - **PlaceOrder Service**: Finalizes the transaction by creating order records, updating inventory, and clearing the cart.

- **Databases**

    - **NoSQL_DBs**:

        - *Product DB*: Stores product metadata for flexible search and display.

        - *Search History DB*, *Recommendation DB*, *Product Review DB*, *Orders DB*: Handle unstructured and semi-structured user activity data.

    - **SQL_DBs**:

        - *Users DB*: Securely stores user credentials and profiles.

        - *Cart DB*: Contains user carts and associated items.

        - *Inventory DB*: Tracks product stock levels and reservations.

Purpose of Architecture

This microservices-based, hybrid-database architecture ensures:

- Scalability across services
- Clear separation of responsibilities
- Optimized storage (structured vs. semi-structured data)
- Secure, role-based access control via central authentication

3. User Requirement Analysis

Functional Requirements

- Search Products
- View Personalized Recommendations
- Place Orders
- Check Order Status
- Write/View Product Reviews

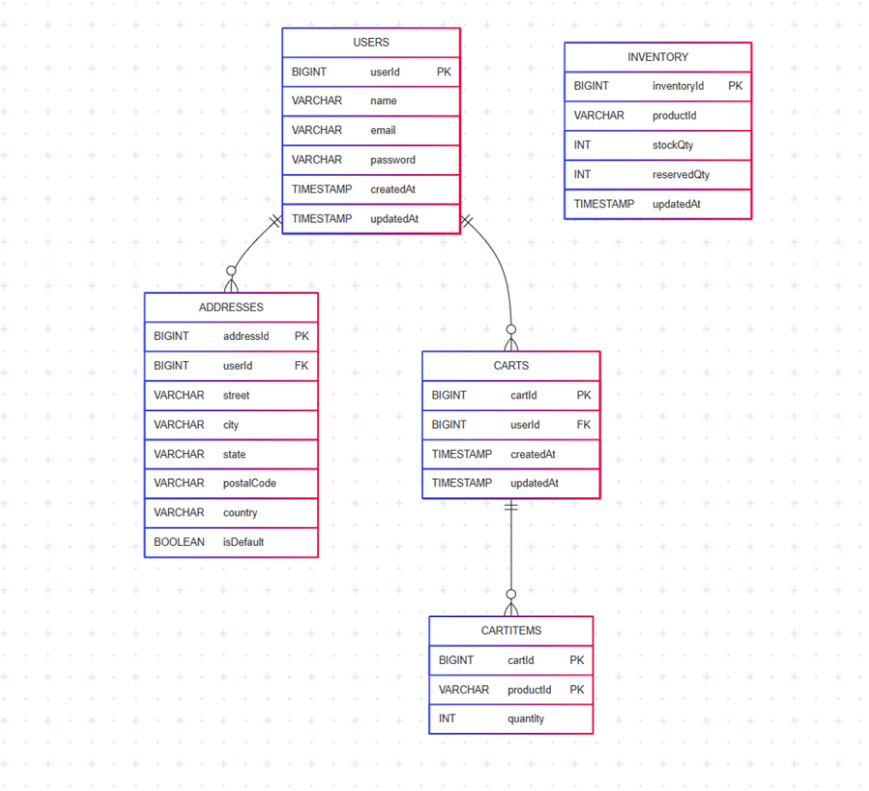
Data Requirements

- User profiles and login credentials
- Product catalog with detailed metadata
- Shopping cart details
- Order history and status updates
- Reviews and ratings
- Search and recommendation history

4. Database Design

4.1 Conceptual Design – Entity-Relationship (ER) Diagram

The following Entity-Relationship (ER) diagram represents the conceptual schema for the relational portion of the E-Commerce database. It captures the core entities and their relationships that facilitate user management, cart handling, inventory tracking, and address associations.



Entity Descriptions

1. USERS

- **Primary Key**: userId
- **Attributes**: name, email, password, createdAt, updatedAt
- **Description**: Stores user account and authentication details. Each user can have multiple associated addresses and shopping carts.

2. ADDRESSES

- **Primary Key**: addressId
- **Foreign Key**: userId → USERS(userId)
- **Attributes**: street, city, state, postalCode, country, isDefault
- **Description**: Captures structured address details linked to each user. One user can maintain multiple addresses, with a flag to indicate the default address.

3. CARTS

- **Primary Key**: cartId
- **Foreign Key**: userId → USERS(userId)
- **Attributes**: createdAt, updatedAt
- **Description**: Represents the active or saved shopping cart for a user. Each cart may contain multiple products.

4. CARTITEMS

- **Composite Primary Key**: (cartId, productId)
- **Foreign Key**: cartId → CARTS(cartId)
- **Attributes**: quantity
- **Description**: Represents individual items within a cart. Links a cart to the products it contains along with their quantities.

5. INVENTORY

- **Primary Key**: inventoryId
- **Attributes**: productId, stockQty, reservedQty, updatedAt
- **Description**: Tracks stock availability for each product, including the total available quantity and the reserved quantity for pending orders.

Relationships Summary

- **USERS → ADDRESSES**: One-to-many (a user can have multiple addresses)
- **USERS → CARTS**: One-to-many (a user can have multiple carts over time)
- **CARTS → CARTITEMS**: One-to-many (each cart can contain multiple items)

This conceptual model ensures clarity in entity roles, promotes data normalization, and sets a foundation for robust relational schema design, supporting key operations such as user login, cart management, and inventory updates.

4.2 NoSQL Data Model (Document-Oriented Design)

In addition to the relational schema, our system uses **NoSQL databases** for unstructured or semi-structured data such as search history, recommendations, reviews, and order logs. These are implemented using a **document model**, where each record is a JSON-like document stored in collections.

1. Product Collection – ProductDB

```
{
  "productId": "P1234",
  "name": "Wireless Earbuds",
  "category": "Audio",
  "description": "Noise-cancelling over-ear earbuds.",
  "price": 79.99,
  "attributes": {
    "color": "Black",
    "batteryLife": "8h",
    "waterResistant": true
  },
  "createdAt": "2025-05-01T00:00:00Z",
  "updatedAt": "2025-05-01T00:00:00Z"
}
```

Purpose: Stores catalog product information for browsing and filtering.

2. Search History Collection – SearchHistoryDB


```
{
  "searchHistoryId": "S3001",
  "userId": 42,
  "terms": "wireless earbuds",
  "filters": {
    "price": [50, 100],
    "brand": ["BrandX", "BrandY"]
  },
  "searchedAt": "2025-05-02T08:30:00Z"
}
```


Purpose: Logs user search behavior for personalization and analytics.

3. Recommendation Collection – RecommendationDB


```
{
  "recommendationId": "REC7007",
  "userId": 42,
  "candidates": [
    { "productId": "P5678", "score": 0.92 },
    { "productId": "P1234", "score": 0.87 }
  ],
  "generatedAt": "2025-05-03T14:00:00Z"
}
```


Purpose: Stores AI-generated product suggestions for each user.

4. Product Review Collection – ProductReviewDB


```
{
  "reviewId": "R5555",
  "userId": 42,
  "productId": "P1234",
  "rating": 4,
  "comment": "Great sound quality and fit!",
  "createdAt": "2025-05-04T10:15:00Z"
}
```


Purpose: Stores customer feedback and star ratings.

5. Orders Collection – OrdersDB


```
{
  "orderId": "O5001",
  "userId": 42,
  "addressId": 7,
  "paymentStatus": "PAID",
  "totalAmount": 149.97,
  "orderDate": "2025-05-04T11:00:00Z",
  "items": [
    { "productId": "P1234", "quantity": 2, "priceAtSale": 49.99 },
```

```
    { "productId": "P5678", "quantity": 1, "priceAtSale": 49.99 }
  ],
  "statusHistory": [
    { "status": "PLACED", "at": "2025-05-04T11:00:00Z" },
    { "status": "SHIPPED", "at": "2025-05-05T09:00:00Z" },
    { "status": "DELIVERED", "at": "2025-05-06T16:30:00Z" }
  ]
}
```


Purpose: Stores full order history and real-time status updates.

These NoSQL collections are used for features requiring flexibility, speed, and scalability. Document-based storage is ideal for dynamic user-generated content and real-time personalization, complementing the structured relational model used for core transactional operations.

4.3 Logical Design – Relational Schema

Tables designed in 3NF:

- **Users**(userId, name, email, password, lastLogin, createdAt, updatedAt)
- **Addresses**(addressId, userId, street, city, state, postalCode, country, isDefault)
- **Carts**(cartId, userId, createdAt, updatedAt)
- **CartItems**(cartItemId, cartId, productId, quantity)
- **Inventory**(inventoryId, productId, stockQty, reservedQty, updatedAt)

Normalization:

- Achieved 3NF.
- Removed repeating groups, partial and transitive dependencies.
- Atomic data fields for integrity.

4.4 Physical Design – Indexing

Indexes are implemented on:

- Users.email (unique login)
- Products.name, Products.category (for faster search)
- Orders.orderDate and Order.status (for real-time tracking)
- Foreign keys such as userId, cartId, productId for quick joins

5. Integrity Design

To ensure data accuracy, consistency, and reliability across the system, our relational database design enforces **three types of integrity constraints**: entity, referential, and domain integrity. These constraints are critical for maintaining the correctness of transactions and the validity of data relationships.

5.1 Entity Integrity

Each table in the relational schema includes a primary key that uniquely identifies each record. These keys are defined as follows:

- Users: Primary Key – userId
- Addresses: Primary Key – addressId
- Carts: Primary Key – cartId
- CartItems: Composite Primary Key – (cartId, productId)
- Inventory: Primary Key – inventoryId

5.2 Referential Integrity

Foreign key constraints enforce valid references between related tables. They are set to **cascade on delete** to ensure dependent records are automatically removed, preserving referential integrity:

- Addresses.userId → Users.userId – ON DELETE CASCADE
- Carts.userId → Users.userId – ON DELETE CASCADE
- CartItems.cartId → Carts.cartId – ON DELETE CASCADE

These constraints ensure that orphaned records are not left in child tables when parent entries are deleted.

5.3 Domain Integrity

Domain constraints ensure that the values in each column are valid and conform to business rules and data types. Examples include:

- Users.email: UNIQUE NOT NULL – Ensures email addresses are unique and mandatory.
- CartItems.quantity: INT NOT NULL CHECK (quantity > 0) – Quantity must be a positive integer.
- Inventory.stockQty: INT NOT NULL CHECK (stockQty >= 0) – Stock quantity must be non-negative.
- Inventory.reservedQty: INT NOT NULL DEFAULT 0 CHECK (reservedQty >= 0) – Reserved quantity must be non-negative and defaults to 0.
- Addresses.isDefault: BOOLEAN NOT NULL DEFAULT TRUE – Indicates whether the address is the default for the user.

These constraints collectively ensure that the data remains accurate, consistent, and secure across all operations within the relational database.

6. Security Design

The system adopts a **role-based access control (RBAC)** model to uphold the principle of least privilege, ensuring that users can only access data and perform operations appropriate to their role.

6.1 Role Definitions

Admin Role (admin_role)

- **Privileges**: Granted full access (ALL PRIVILEGES) to the entire ecommerce_db.
- **Responsibilities**:
    - Perform schema modifications
    - Manage user accounts and roles
    - Execute system backups and recovery operations
    - Audit system activity

Customer Role (customer_role)

- **Users**: SELECT, UPDATE
- **Addresses**: SELECT, INSERT, UPDATE
- **Carts**: SELECT, INSERT, UPDATE
- **CartItems**: SELECT, INSERT, UPDATE
- **Inventory**: No access

These permissions ensure that customers can manage their profiles, carts, and addresses but are restricted from accessing administrative or inventory-level data.

6.2 Key Security Benefits

- **Separation of Duties**: Prevents privilege escalation by strictly limiting customer access to sensitive operations.

- **Auditability**: All actions are role-bound, simplifying logging and compliance monitoring.

- **Security Hardening**: Centralized control reduces the attack surface for unauthorized access.

6.3 Security for NoSQL Databases

NoSQL components (e.g., product reviews, search history, recommendations) are protected via **application-layer security**:

- **API Gateway & Authentication**:

    - All database access requests pass through an API layer secured by **OAuth2** or **JWT (JSON Web Tokens)**.

    - This layer authenticates the user's identity and verifies their role before granting access to any document or resource.

This comprehensive security design ensures that the database system is robust against unauthorized access while maintaining usability for legitimate users.

7. Application Functional Design

The application provides a set of core functionalities that support essential e-commerce operations. Each function is designed as an independent service and interacts with specific database components (SQL or NoSQL) to fulfill business requirements.

1. GetRecommendations(UserID)

Purpose: Retrieve the top 10 recommended product IDs for a user.
Data Source: Reads from the **Recommendation Database** (RecDB) – *NoSQL*.
Behavior: Returns personalized recommendations based on prior activity and preferences.

2. Search(SearchString, UserID)

Purpose: Return a list of products matching the search query.
Data Source:

- Reads from the **Product Database** – *NoSQL*

- Logs the search terms and filters to the **Search History Database** (SearchHist) – *NoSQL*
  Behavior: Enables keyword-based catalog exploration with contextual filters.

3. AddToCart(UserID, ProductID, Qty)

Purpose: Add or update a product entry in the user's shopping cart.
Data Source: Writes to the **Cart Database** (CartDB) – *SQL*.
Behavior: Inserts or modifies the line item with the given quantity for the user's cart.

4. PlaceOrder(UserID, OrderID, AddressID, PaymentStatus)

Purpose: Finalize the purchase of items in the user's cart.
Data Flow:

- Reads from the **Cart Database** (CartDB) – *SQL*
- Writes to the **Order Database** (OrderDB) – *SQL*
- Updates inventory in the **Inventory Database** (InvDB) – *SQL*
- Deletes processed cart entries
  Behavior: Executes the transaction, creates a new order, adjusts stock levels, and clears the cart.

5. CheckOrderStatus(OrderID)

Purpose: Retrieve the current status of a user's order.
Data Source: Reads from the **Order Database** (OrderDB) – *SQL*.
Behavior: Provides real-time updates on order progress (e.g., placed, shipped, delivered).

6. WriteReview(UserID, ProductID, Rating, Comment)

Purpose: Submit or retrieve product reviews and feedback.
Data Source: Writes to and reads from the **Product Review Database** (ReviewDB) – *NoSQL*.
Behavior: Enables users to review products they've purchased and view others' feedback.

Each of these services is implemented in a modular fashion and communicates with the respective data layers to maintain separation of concerns, scalability, and ease of maintenance.

8. Database Implementation & Maintenance

Tools Used

- **Relational DBMS**: MySQL
- **NoSQL DBMS**: MongoDB
- **Integrator**: Python
- **Authentication**: OAuth2 / JWT

Experimental Data

- Synthetic test data inserted into all tables
- Validated CRUD operations on user profiles, carts, and order processing

9. System Running Results and Analysis

(Add screenshots of the following operations from the running database UI or backend logs)

10. Conclusion

This project successfully demonstrates the design and implementation of a robust, scalable database system tailored for an E-Commerce platform. Through a combination of relational (SQL) and document-based (NoSQL) models, the system effectively supports key functionalities such as user management, product search, personalized recommendations, cart operations, order processing, and customer reviews.

We achieved:

- A fully normalized relational schema up to the third normal form (3NF)
- Clear entity relationships and data integrity through comprehensive constraints
- A secure, role-based access control model ensuring proper authorization and data protection

- An application-layer architecture that integrates modular services with efficient database interactions

- A hybrid storage approach leveraging NoSQL for flexible, high-volume user data such as recommendations and reviews

This design provides the foundation for a scalable, secure, and user-centric E-Commerce system, capable of handling millions of users and high transactional throughput. Future enhancements may include AI-driven analytics, real-time inventory syncing, and advanced data caching for performance optimization.